UNIVERSAL PROGRAM GENERATOR

UPG

THIS SOFTWARE LICENSE AGREEMENT IS YOUR PROOF OF LICENSE TO USE THE PERCON SOFTWARE PROGRAM KNOWN AS THE UNIVERSAL PROGRAM GENERATOR IN EITHER THE FULL DEVELOPMENT SYSTEM VERSION OR THE RUNTIME VERSION (THE "SOFTWARE") UNDER THE TERMS DESCRIBED BELOW. IF YOU DO NOT AGREE WITH THE TERMS OF THIS AGREEMENT FOR ANY REASON, YOU MAY RETURN THE SOFTWARE IMMEDIATELY TO THE LOCATION WHERE YOU PURCHASED IT FOR A REFUND OF THE PURCHASE PRICE OF THE SOFTWARE.

THIS SOFTWARE MAY ONLY BE USED WITH A COPY-PROTECT "SECURITY KEY" SOLD AND PROVIDED TO YOU WITH THE SOFTWARE. A SEPARATE SECURITY KEY AND A SEPARATE LICENSE FOR THE APPLICABLE FULL DEVELOPMENT SYSTEM OR RUNTIME VERSION OF THE SOFTWARE MUST BE PURCHASED FOR EACH COMPUTER ON WHICH THE SOFTWARE OPERATES, INCLUDING EACH COMPUTER WHICH WILL INTERFACE WITH A PORTABLE BAR CODE READER. DISABLEMENT OF THE SECURITY KEY SO THAT THE SOFTWARE WORKS WITHOUT THE SECURITY KEY IS STRICTLY PROHIBITED.

PERCON, Incorporated ("PERCON") grants to you a nonexclusive, nontransferable license to use this copy of the Software and accompanying documentation according to the following terms:

LICENSE

You may:

(1) use the "Full Development System" version of the Software provided to you for use with a single computer only; or
(2) use the "Runtime System" version of the Software provided to you for use with a single computer only, to allow the computer to interface with a portable bar-code reader;
(3) install the Software only on the number of computers for which you have purchased a license, each of which require a Security Key;
(4) make one copy of the Software in machine readable form solely for backup purposes, provided that you reproduce all proprietary notices on the copy, including copyright, trademark and any other similar notice of PERCON's proprietary interest in the Software;
(5) physically transfer the Software from one computer to another, provided that the Software is installed on only one computer at a time.

You may not:

(1) use the Software on more than one computer;
(2) copy and distribute the source code to the Software;
(3) modify, translate, or copy the accompanying documentation to the Software;
(4) rent, transfer or grant any rights in the Software or accompanying documentation in any form to any person without obtaining a license from or the prior written consent of PERCON;
(5) remove any proprietary notices, labels, or marks on the Software and accompanying documentation; or
(6) use the communications utilities (UPGXFER.EXE or XFERSMBL.EXE) without purchasing a license for the Runtime version of the Software for each computer that will interface with a portable bar-code reader.

This license is not a sale. Title and copyrights to the Software, accompanying documentation and any copy made by you remain with PERCON. Unauthorized copying of the Software or the accompanying documentation, or failure to comply with the above restrictions, will result in automatic termination of this license and will make available to PERCON other legal remedies.

30-DAY GUARANTEE; DISCLAIMER

If the Software is inoperable due to any defects in the media on which the Software resides, then you may return the Software to the location where you purchased the Software within the 30-day period from the date of your invoice. PERCON's entire liability and your exclusive remedy under this guarantee (subject to your returning the Software to the location where you purchased the software with a copy of your receipt) will be, at PERCON's option, either (a) return of the price paid or (b) repair or replacement of the Software. This guarantee does not apply to the Software if the failure of the Software has resulted from accident, abuse or misapplication. Outside of the United States, these remedies are not available without proof of purchase from an authorized non-U.S. source. In the case of your return of the Software for any reason, you will delete any copy of the Software on your computer, and any other copy in object code or source code form, and you further agree not to use the Software until a repaired or replaced copy of the Software is returned to you by PERCON.

EXCEPT FOR THE ABOVE GUARANTEE, PERCON MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR IN ANY COMMUNICATION WITH YOU, AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTY OF NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. PERCON DOES NOT WARRANT THAT THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE.

LIMITATION OF LIABILITY

IN NO EVENT WILL PERCON BE LIABLE FOR ANY DAMAGES, INCLUDING LOSS OF DATA, LOST PROFITS, COST OF COVER OR OTHER SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES ARISING FROM THE USE OF THE SOFTWARE OR ACCOMPANYING DOCUMENTATION, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY. THIS LIMITATION WILL APPLY EVEN IF PERCON OR AN AUTHORIZED RESELLER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. YOU ACKNOWLEDGE THAT THE LICENSE FEE REFLECTS THE ALLOCATION OF RISK. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

TECHNICAL SUPPORT

PERCON will provide limited technical support at no cost to you for 30 days from the date of your purchase. This technical support does not include assistance in developing applications or assisting you with any C Code, including code hooks or external functions.

GOVERNMENT RESTRICTED RIGHTS LEGEND; INTERNATIONAL USE

Use, duplication or disclosure by the United States Government is subject to restrictions of Restricted Rights for computer software developed at private expense as set forth in FAR Sec. 52.227-19 or DOD FAR Supplement Sec. 252,227-7013(c)(1)(ii), and successor thereof, as applicable. If you are sublicensing or using the Software outside of the United States, you must comply with the applicable local laws of your country, U.S. export control laws, and the English language version of this Software License Agreement.

GENERAL

This Agreement is governed by the laws of the State of Oregon, without reference to conflict of laws principles. This Agreement is the entire agreement between us and supersedes any other communications or advertising with respect to the Software and accompanying documentation. If any provision of this Agreement is held invalid, the remainder of this Agreement will continue in full force and effect. If you have any questions, please contact in writing: PERCON, Incorporated, 1720 Willow Creek Circle, Suite 530, Eugene, Oregon 97402.

Technical Support

Percon provides free technical support for 30 days via telephone, fax, BBS, e-mail and our home page on the World Wide Web.

- ❑ Phone: (541) 344-1189
- ❑ Fax: (541) 344-1399
- ❑ BBS: (541) 334-4413
- ❑ E-mail: tech@percon.com
- ❑ Internet: http://www.percon.com

Extended Warranty / Maintenance

Optional extended warranty or “maintenance plans” may be purchased to provide support after the initial warranty period has expired. Two types of maintenance plans are available: Standard Maintenance Plan and the Premium Maintenance Plan. Contact Percon or your sales representative for more information and pricing.

The information contained in this document is subject to change without notice.

A

B

1 UPG Overview

Welcome to UPG

Percon’s Universal Program Generator (UPG) is a revolutionary product that allows a non-programmer to create a single data collection application that runs on many different brands of portables. Traditionally, running the same application on multiple brands of portables required a C programmer to develop and maintain many separate programs. With UPG, you can now develop one program to handle multiple screen sizes, multiple languages, and multiple portables, without writing a single line of “C” code. With UPG, you can build a complex data collection application in a matter of hours, whereas it would take an experienced C programmer days or possibly weeks to develop the equivalent application.

Many different levels of users will find that UPG provides an extensive array of tools to build the portable applications they want and need. UPG is simple enough for the novice user to become productive within hours, yet flexible enough to allow experienced “C” programmers to develop advanced applications by including their own code using UPG’s Code Hooks.

UPG includes the following features:

- ❑ Supports a Variety of DOS Portables
- ❑ Generates Standard ANSI C Code
- ❑ Reusable Tools for Enhanced Productivity
- ❑ Keyed File Access for Fast Lookups
- ❑ Integrated Compiler
- ❑ Compatible with Borland and Microsoft Compilers
- ❑ Conditional Branching
- ❑ Extensive Data Validation Features
- ❑ Advanced Data Manipulation Capabilities, Including Parsing
- ❑ Support for Multiple Languages
- ❑ Support for Multiple Screen Sizes
- ❑ Code Hooks for Embedding Developer Source Code
- ❑ Integrated Spell Checker
- ❑ Modem Support
- ❑ ZModem, Host Mode Communications

What is UPG?

UPG is a 32-bit Windows application designed to help developers and users of portable data collection units build custom solutions for their data collection needs. UPG generates standard ANSII “C” source code and compiles to a standard EXE.

The UPG application is a complete package that includes everything you need to build a portable data collection application, including a C compiler; for those users who wish to, UPG can be configured to use either the Borland 4.5 or Microsoft Visual C++ 1.5 (or greater) “C” compiler.

System Requirements

UPG will run with the minimum requirements listed below, but for maximum performance, install UPG onto a computer that contains a higher speed processor and more memory than the minimum requirements.

Hardware

- □ Processor - 80486/50MHz (Pentium recommended)
- □ Memory - 16MB (24 recommended)
- □ Hard Disk - 10MB
- □ Video - VGA

Software

- □ Windows 95
- □ Windows NT 3.51 or greater

Supported Portables

In addition to supporting the Percon Falcon, UPG Version 1.0 supports several platforms from Symbol and Intermec. (You will need the appropriate docking station or communications cable for your portable data collection unit to work with UPG; if you do not have the necessary portable accessories, contact your portable data collection supplier.)

Tip: Visit Percon’s website at http://www.percon.com to get the latest on supported portables, as the list of UPG supported portables will continually increase.

Percon

- □ Falcon

Symbol

- □ 3100
- □ 3805

Intermec

- □ Janus 2010
- □ Janus 2020

UPG Interface Conventions

The UPG interface consists of three distinct areas of operation: the Menus/Toolbars, the ToolBox pane, and the Program pane. Many of the options available on the Toolbar or in the menus are also available when you press the right button (right-click) on your mouse; such options may include adding a specific tooltype, copying and pasting a tool, or adding controls (the available pop-up menu selections reflect the operations that can be performed in your present UPG location).

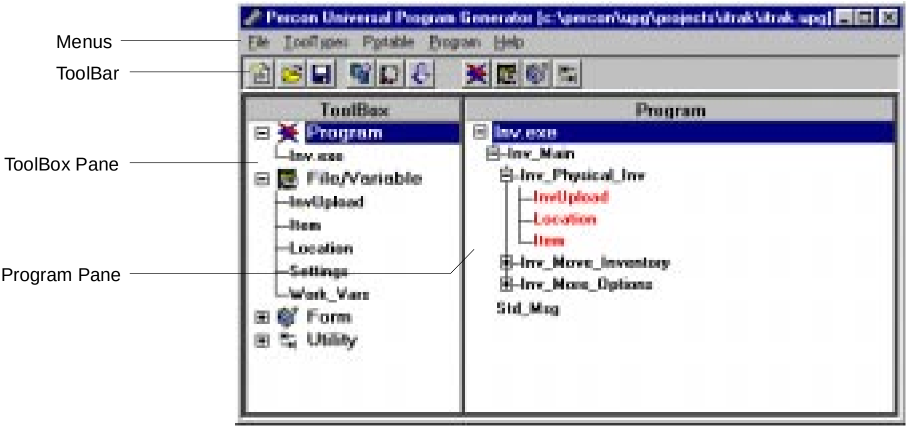


Figure 1-1: UPG Work Environment

UPG Terminology

Before you dive into the world of UPG development, now might be a good time to define some terminology that you will no doubt encounter within this manual and within the UPG programming environment.

ToolBox - A ToolBox (left pane) stores a collection of *tools* (see definition below) that combine to create individual portable applications; the toolbox is stored as a .UPG file. A toolbox may contain more than one portable application which may be helpful when developing multiple applications with similar feature sets. The items stored in a toolbox are listed in the ToolBox pane of your UPG work space.

ToolType - Tooltypes are the UPG components used to create unique tools, all of which are stored in a toolbox file (.UPG). The available UPG tooltypes are Program, File/Variable, Form, and Utility.

Tool - A tool can simply be defined as a program operation that performs a specific task. For example, the **Inv_Physical_Inv** multipage sequence in **Figure 1-1** is a UPG tool. A portable application is built using the tool developed from the different *tooltypes*. Think of tools as the individual building blocks used to construct a portable application.

Control - Controls are the user interface components you use to create a portable application. The five available control types are Text (text labels, titles), Hotkey (function keys), Input (entry) Field, Display Field, and Alias. Controls can only be used with tools created using the Form tooltype.

Code Hooks - Code Hooks are provided by the UPG application to allow “C” programmers to insert (embed) their own code within the UPG generated code; this allows experienced users to create an advanced custom solution.

UPG Action Components - UPG provides Action components to add powerful features to your portable applications. You can add loops, conditional IF-THEN-ELSE structures, and calculations to your portable application using UPG Action components.

File - When you collect data with your portable data collection unit (or any computer for that matter), the data that you have collected is written (stored) to a file; your target file may be written to a physical disk or to your computer’s memory. Files are made up of individual fields that store separate pieces of information. Think of a file as a warehouse and a field as an individual storage bin.

Variable - Computers use Variables as a temporary storage location for information. Unlike a file, information stored in a variable will be lost when the application using the variable is terminated. Variables are handy for processing information that will be used during the application session, but are not permanently saved.

Compiler - A Compiler is a utility that converts standard ANSI C Code into an executable file (.EXE). When you launch an application, it is usually an executable file that you are running.

ANSI C Code - Code that complies with the guidelines established by the American National Standards Institute. The source code that is generated by UPG meets ANSI standards.

Menus/ Toolbars

UPG tasks can be performed by selecting them from the menu bar at the top of the UPG work space; many of the most commonly used items are also available on the UPG toolbar (see **Figure 1-2**) to make repetitive tasks more accessible. The following paragraphs will briefly detail the items available in each of the menus.

File - The File menu allows the user to create, open, and save toolboxes. The File menu also allows the user to configure the UPG application to meet their individual needs. Specifically, the Options menu selection allows you to specify application defaults, such as reloading the last toolbox on startup, and compiler settings.

ToolType - The ToolType menu allows you to create a new tool by selecting the desired tooltype from the menu; selecting a tooltype from the menu performs the same function as double-clicking on a tooltype in the ToolBox pane. The ToolType menu also has a Copy Tool and a Paste Tool selection; as the names suggest, these copy and paste selected tools.

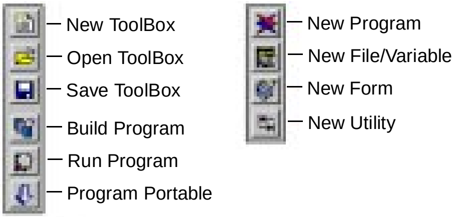


Figure 1-2: Toolbar Button Assignments

Portable - This menu allows you to specify the portable model and language version of your portable application that UPG will generate code and/or compile. The Portable menu also contains the commands to program your portable with the UPG generated application, as well as a File Transfer Manager to transfer data files between your Host PC and portable data collection unit.

Program - The Program menu contains the commands to build your program and add external files to your UPG generated application. You can also run your application in a DOS window by selecting **Run**.

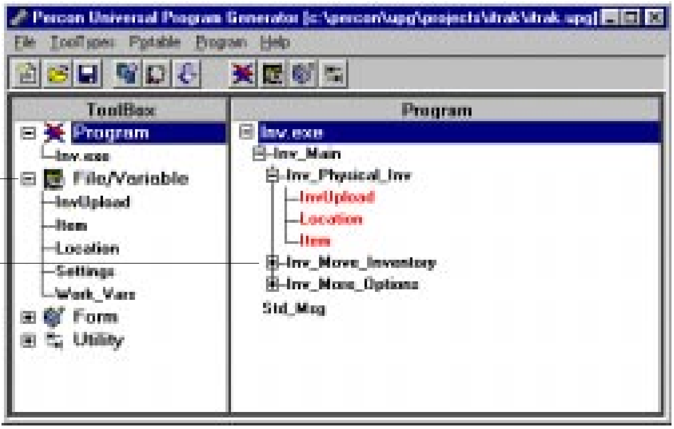
Help - This menu allows the user to access the on-line help and tutorials provided with the UPG application.

Note: Context sensitive help is available throughout UPG; simply press <F1> or press the **HELP** button to view the available help. When you receive context sensitive help, steps on how to complete your current task will appear. If detailed information is available, press the More Info link at the top of the context sensitive help screen to view the detailed information.

ToolBox Pane

The UPG ToolBox stores all of the available tooltypes and their unique tools. A toolbox file (.UPG) may contain more than one portable application, which may be very useful if you have similar portable applications that share similar components. For example, if you have a product line of portable applications, creating all of the portable applications in one toolbox may drastically reduce development time, since many of these products probably have shared components.

To view all tools of a specific tooltype, click on the expand button; the tooltype will expand to list all defined tools (see **Figure 1-3**). You will also notice that when you select a specific function in the UPG ToolBox. The corresponding function in the Program pane will be highlighted if the function is included in the currently selected portable program.



Press to Collapse

Press to Expand

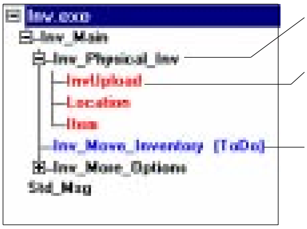
Figure 1-3: Using the indentation and relationship line conventions, you can easily see the linked relationship between the individual tools that form your UPG portable application.

You can use a variety of different methods to add a new tool to the toolbox: by selecting the desired tooltype from the ToolType menu, clicking on the corresponding tooltype icon in the UPG toolbar, or right-clicking on the desired tooltype in the ToolBox pane and selecting Add from the pop-up menu. The quickest method to create a new tool, however, is to simply double-click on the desired tooltype in your ToolBox.

Program Pane

The Program pane is the portion of the UPG work environment that you use to construct a UPG portable application. The UPG Program pane uses several conventions to help you identify the different functions that make up your application; the most obvious convention is the use of indentation and relationship lines.

The indentation of each program item indicates the function's place in the program hierarchy. For example, in **Figure 1-4**, the **Inv_Physical_Inv** form is indented 1 level under the **Inv_Main** form; this indentation example shows that the **Inv_Physical_Inv** multipage sequence form is a sub-form of the **Inv_Main** form.



Inv_Physical_Inv is a multipage sequence that is attached to the **Inv_Main** menu.

InvUpload is a file attached to the **Inv_ Physical_Inv** tool.

Inv_Move_Inventory (To Do) is a multipage sequence that is attached to the **Inv_Main** menu; it is labeled with a (To Do) because it was called in the main menu, but it has not been defined.

Figure 1-4: The Program pane uses indentation and relationship lines to help you identify the connections between the different tools of your portable application.

In addition to the indentation conventions, relationship lines are drawn between inter-related functions. In **Figure 1-4**, you will notice how the line stemming from the **Inv_Main** form branches off and intersects the sub-forms **Inv_Physical_Inv**, **Inv_Move_Inventory (To Do)**, and **Inv_More_Options**. **Figure 1-4** graphically shows that the three sub-forms are attached to the **Inv_Main** form.

A (To Do) denotes that a tool has been referenced, but is currently not defined. If you have a (To Do) in your Program pane, you can convert it to a defined tool by simply double-clicking on the (To Do) and selecting the tooltype you would like to use; for instance, you can double-click on a (To Do) and you can either select **Form** or **Utility**. After making your selection, you will be ready to build your tool (please see Chapter 4, Using UPG for more information).

Tool Relationship

There is a very close (and dependent) relationship between the tools that exist in the ToolBox pane and the tools that exist in the Program pane. Tools that are only in your ToolBox are *unlinked*, whereas tools that are in your Program pane, are *linked*. So, what does that mean to you?

Notice how the applications are built visually in the Program pane of the UPG work environment (see **Figure 1-6**); the individual tools contained in the ToolBox are linked together in the Program pane to form a complete structure. Consider the following analogy; *unlinked* tools are the individual building blocks in the ToolBox used in UPG development; combine or *link* the individual building blocks in the Program pane to form a complete structure or, in this case, a portable application.

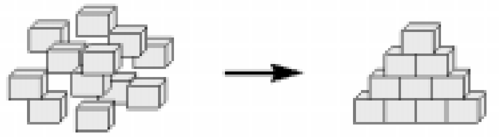
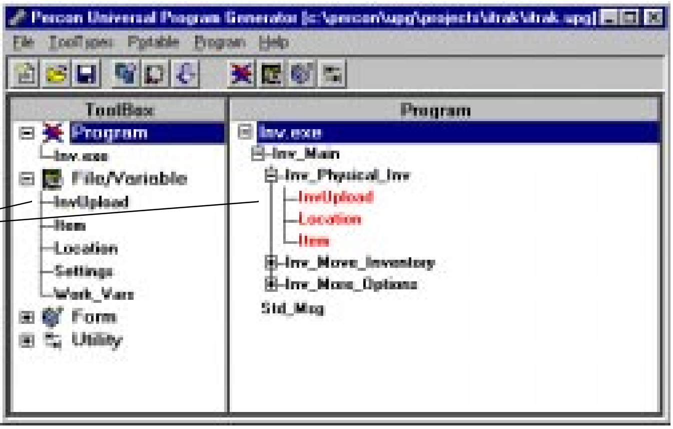


Figure 1-5: Just like one might use individual building blocks to build a pyramid, use (link) individual tools to build complete portable applications.

Unlinked tools that exist in your ToolBox are waiting to be used in your portable application; these tools are reusable, and can therefore be used multiple times within the same application. Note however, a tool may remain unused in the ToolBox pane without ever being used in the Program pane; this will not affect your UPG generated application.

When you make a modification to a tool in your ToolBox, the tool in your program pane will have the same changes made to it. Conversely, if a change is made to a tool in the Program pane, the change will also be made to the tool in your ToolBox.



Though these tools are in different panes, any change made to one tool will change the tool in the other pane.

Figure 1-6: You can easily identify what tooltype a particular tool was created from by the color of the tool. The tool will be same color as the tooltype in the ToolBox pane.

2 INSTALLATION & SETUP

Install Universal Program Generator

UPG has a simple to use installation utility that will guide you through the installation routine.

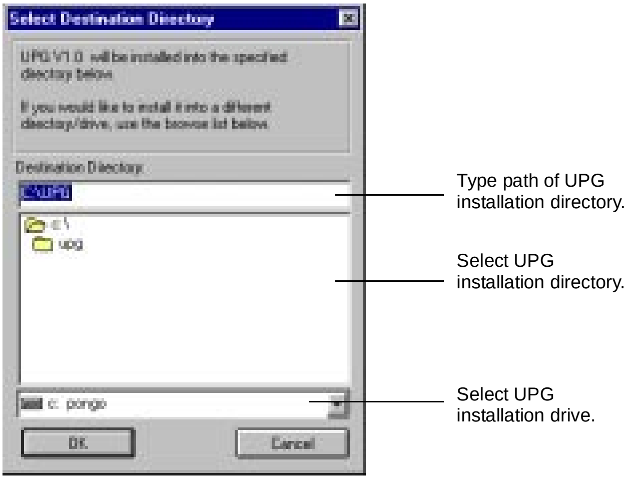
1 **Attach the security key to your computer’s parallel (printer) port.**

2 **Select RUN from the Windows START menu.**

3 **Type `a:setup` in the OPEN field and press the OK button**
If your 3 1/2 floppy disk drive is not your a: drive, please substitute the appropriate drive letter; for example, **`b:setup`**.

4 **When you have completed reading the Welcome screen, press the OK button to continue your UPG installation.**

5 **Select the UPG destination directory (see fig. 2.1).**
UPG installs to the `C:\UPG` directory by default; select or create another directory if desired.



Select Destination Directory

UPG V1.0 will be installed into the specified directory below.

If you would like to install it into a different directory/drive, use the browse list below.

Destination Directory: C:\UPG

Type path of UPG installation directory.

Select UPG installation directory.

Select UPG installation drive.

Figure 2-1: Select UPG Install Directory

6 **Select the Destination where you want to install the UPG startup icon.**
The UPG startup icon will be placed in the UPG program group by default; select or create another startup group as desired.

7 **Press OK when prompted to reboot your computer.**
It is necessary to reboot your computer to properly complete the installation of UPG; if you do not reboot your computer, UPG may not work properly.

8 **Launch the Universal Program Generator.**
When you have completed steps 1-6, you are ready to run UPG:

□ Press the Windows START button.

□ Select PROGRAM.

□ Select Universal Program Generator (or whatever program group you specified in step 6).

□ Select UPG.

Note: If you ever have difficulties trying to launch UPG, open a DOS window and open the directory where UPG was installed (in most cases, `c:\upg`). At the DOS prompt, type `UPG /reset`. This will return UPG to the original installation settings, and in most cases will allow you to launch UPG.

Uninstall UPG

If at any time you need to uninstall the Universal Program Generator from your computer, please complete the following steps:

1 **Press the Windows START button.**

2 **Select PROGRAM.**

3 **Select Universal Program Generator (or whatever program group you specified during installation).**

4 **Select UNINSTALL UPG.**

5 **Press the AUTOMATIC button to begin uninstalling UPG.**
Pressing the AUTOMATIC button will remove all of the installed components of UPG from your computer.

6 **You may need to reboot your computer to finish completing the uninstallation process.**
You may notice that the UPG directory still remains; this is because any projects that you may have built using UPG may be saved in the \Projects subdirectory of UPG. If this is the

case, the UPG directory will not be removed because the project you created was not one of the original installation files; the directory is preserved so you don't lose any of your existing projects.

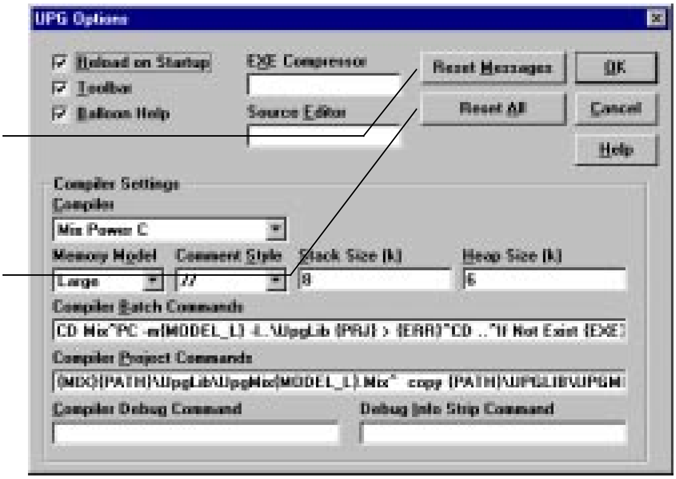
Configure UPG Options

Percon's Universal Program Generator is ready to run immediately after you complete installing UPG, but there are several optional settings you can configure to meet your individual requirements.

To modify your UPG settings, please complete the steps below:

1 **Select Options from the File menu.**

2 **Modify the settings according to the descriptions below.** Each individual setting is discussed in detail in the Work Environment and Compiler Settings sections that follows this series of steps.



UPG Options

☑ Reload on Startup
☑ Toolbar
☑ Balloon Help

EXE Compressor:

Source Editor:

Compiler Settings

Compiler: Mix Power C

Memory Model: Large

Comment Style: /?

Stack Size (k): 8

Heap Size (k): 6

Compiler Batch Commands: CD Mix^PC -m{MODEL_L} -I..\UpgLib {PRJ} > {ERR}^CD ..^If Not Exist {EXE}

Compiler Project Commands: {MIX}{PATH}\UpgLib\UpgMix{MODEL_L}.Mix^ copy {PATH}\UPGLIB\UPGM

Compiler Debug Command:

Debug Info Strip Command:

Press Reset Messages to restore or re-enable messages that you previously set to "Don't Show Again."

Resets UPG to the original installation defaults; all compiler settings will be set to the original.

Figure 2-2: The above settings are the defaults that are set when UPG is installed onto your computer.

Warning: It is recommended that only experienced C programmers modify the UPG compiler settings; UPG is installed with all of the necessary compiler commands optimized for integration with UPG.

3 **Press the OK button when you have completed modifying your UPG settings.**

Work Environment

The work environment settings allow you to configure several features of the UPG user interface. The settings contained in this section are optional and are made available so that you can modify the user interface to fit your work habits.

Reload on Startup: Enable (check) this checkbox to automatically load the last open application when UPG is launched.

Toolbar: Enable this checkbox to display the toolbar above the UPG work environment; the toolbar provides shortcuts to many popular functions (see **Figure 2-3**).

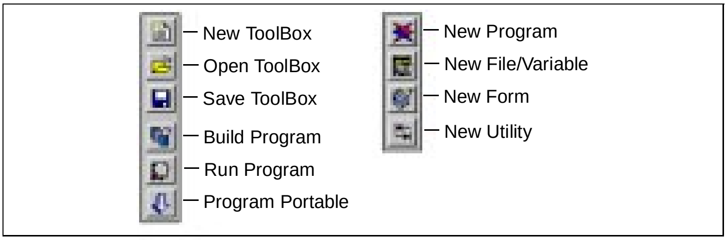


Figure 2-3: Toolbar Button Assignments

Balloon Help: Enable this checkbox to display balloon help when your mouse is positioned over the toolbar; balloon help will allow you to identify the function of the individual buttons listed in **Figure 2-3**.

EXE Compressor: UPG includes WWPACK; this EXE compressor is automatically used by UPG to conserve space on your target portable.

If you need to specify a different EXE compressor, please enter the executable name of the compressor in this field. If your EXE compressor is not included in the `Path` statement of your `AUTOEXEC.BAT`, you will need to enter the fully qualified path of your executable (i.e. `c:\compress\compress.exe`).

An EXE compressor is very useful when you have built a large application (greater than 150K). Portables are typically limited in the size of applications they can run; using an EXE compressor will allow you to run larger applications that might not otherwise operate correctly in an uncompressed state.

Source Editor: UPG uses Windows Notepad as the default source editor. If you have an editor you would like to use with UPG, enter the executable in the Source Editor field. If your editor is not included in the `Path` statement of your `AUTOEXEC.BAT`, you will need to enter the fully qualified path of your executable (i.e. `c:\editor\editor.exe`).

Compiler Settings

The compiler settings allow you to alter the commands sent to the attached C compiler to meet your individual needs. You may want to consult your compiler documentation or a C programming reference for more information on the available compiler settings.

Warning: It is highly recommended that only experienced C programmers modify these settings, as an incorrect setting may cause UPG compile errors.

Note: The compiler settings that are installed by default have been optimized for integration with UPG. The default compiler settings for the Borland and Microsoft compilers have also been optimized if you choose to use one of those compilers.

Compiler: Select the compiler you will be using to compile your UPG generated applications. Mix Software’s Power C is selected by default (this is the integrated compiler that is included with your copy of UPG). If you have licensed copies, you may also select the Borland C++ 4.5 or Microsoft Visual C++ 1.5 compiler.

Memory Model: Select the memory model that you would like the attached compiler to use when compiling your executable application. The default selection, Medium, has been selected as the optimum setting for the integrated Power C compiler. If you attach another manufacturer's compiler, consult your compiler documentation to determine the best memory model for your individual application.

Note: If you selected Borland as your compiler, you can also select Overlay as a valid Memory Model. When you use an Overlay memory model, your object code will only be loaded into your portable's memory and executed when needed. Overlays allow you to load large programs onto portables that do not have the available resources to execute the entire application (for example, you may need to use an Overlay memory model when trying to program Symbol portables with an application greater than 175k).

Comment Style: Select the comment style that you would like used when UPG places comments in the generated source code; the // comment style is selected by default.

Stack Size (k): Specify the amount of memory that you would like allocated to application variables. The default value has been optimized for portable data collection applications.

Heap Size (k): Specify the amount of memory that you would like reserved for application use. The default value has been optimized for portable data collection applications.

Compiler Batch Commands: Creates the batch file that runs the compiler, linker, or make utility that is used to create your executable. This batch file is named UPGBUILD.BAT and is stored in the UPG directory. The Compiler Batch Commands use the symbols listed in Table 2-1 on page 2-8 to dynamically assign information appropriate to the current project.

Compiler Project Commands: Creates the project file for your application. A caret (^) denotes a hard return (end of line). The Compiler Batch Commands use the symbols listed in Table 2-1 on page 2-8 to dynamically assign information appropriate to the current project.

Symbol	Description
{EXE}	Fully qualified path name of the EXE file we're generating
{PRJ}	Fully qualified path name of the project file generated by the Build Project Command and generally but not necessarily used by the Compile command
{ERR}	Fully qualified path name of the error file that UPG will display if the build seems to fail (Failure is detected by the build window disappearing, and the EXE not existing)
{STACK}	Stack size as entered in the Configuration window as a decimal number.
{STACK$}	Stack size displayed as a hexadecimal number (used by Microsoft linker)
{HEAP}	Heap size as entered in the Configuration window as a decimal number.
{HEAP$}	Heap size displayed as a hexadecimal number (used by Microsoft linker)
{PATH}	Path where UPG.EXE resides. Note that UPG always changes to this directory before starting the UPGBUILD.BAT.
{SOURCE}	A list of all source files (with .C extension) for this project with one file per line. Each file is a fully qualified path name.
{OBJ}	A list of all source files with the extension changed to .OBJ.
{MIX}	A list of all source files with the extension changed to .MIX (for the Mix Power C compiler).
{MODEL_L}	The first letter of the memory model as a lower case letter (e.g. Large memory model produces l)
{MODEL_U}	The first letter of the memory model as an upper case letter (e.g. Large memory model produces L)
{PROGLIB}	A list of all LIBRARY files added to the current program with one file per line. The files appear exactly as they show up in the program pane.
{PROGLIBMIX}	A list of all LIBRARY files added to the current program with one file per line and the extension of all files changed to .MIX.

Table 2-1: Use the symbols contained in this table to build the Compile Batch Commands and the Compile Project Commands; information appropriate to the currently select UPG project will be passed to the symbols. This will allow you to use the same compile commands between multiple UPG projects.

Compiler Debug Command: Specify the executable of the debugger you are using; make sure that this executable is in the Path statement of your Autoexec.bat file or enter a fully qualified path name in this field (i.e. - `c:\debug\debug.exe`).

Debug Info Strip Command: Enter the command for your debugger to remove the Debug Info placed in your EXE file; this

Debug Info will be removed when your program your portable. It is especially important to remove Debug Info when you are working with large applications (greater than 150K).

3 PROGRAMMING METHODOLOGY

Introduction to UPG Programming Methodology

When you begin a new UPG project, it is strongly encouraged that you take the time to properly determine the requirements of your portable data collection application. After the requirements have been determined, you will be in a better position to build a comprehensive project plan. The following sections will help you plan your project.

Portable Application Purpose

Quite simply, you need to definitively state the objective of your portable application; if necessary, write this objective on a piece of paper and answer the questions below. Verify that the answers to these questions adhere to the objective you have stated. If any of your answers do not adhere to your objective, then you will need to re-evaluate your objective and modify as needed or change your answer to meet your objective's requirements.

Ask Yourself

1. **What information do you want to collect?**
2. **In what order is the data usually collected? (Make a flowchart if this will help you answer the question.)**
3. **In response to question 1, are all of the pieces of collected data essential to meet your program's objective?**
   Could the objective be better met by using related data? (If you collect an item number, you really need to collect a description also; this description is probably already related to the same item number in one of your existing databases.)
4. **Which pieces of the collected data are required?**
5. **Are any of the pieces of collected data required to be a unique value?**
6. **How will barcodes be used in the data collection process?**
7. **With what applications will you be sharing data?**
8. **Will you want any of your collected data to be validated?**

Validation checks to see if the collected data exists in another file. For example, you may want to validate the user ID's or item numbers to ensure consistent (valid) input.

9 **What portable platforms will you be developing for?**
UPG has the capability to develop applications for multiple platforms. When determining your application requirements, consider the requirements of your target portables, particularly the display size.

Although it may sound like a tedious task, take the time to carefully consider your objective and the answers to the questions above. The time and effort you put into this process may save you much wasted time in re-thinking and modifying your portable application.

Determine Storage Structure

When you have defined the purpose of your portable application and answered the questions above, you are now ready to begin considering the data storage facilities that will be required by your portable application. As with defining the purpose of your portable application, there are several questions to address before physically defining your storage structure.

Ask Yourself

1 **Will you be transferring (uploading) your collected data to an existing application?**
If you answered yes, what import requirements does that application have? (Consider field delimiter, data type, field lengths, required/unique values, and field order.)

2 **Do you want or need to store your collected data in multiple files?**

When you have answered the above two questions, it is time to get out some graph paper (or use a spreadsheet application) and begin defining your storage structure. The components of your storage structure should include field order, field name, data type, maximum field length, required (is the field required?), and unique (does the value need to be unique?). Model your worksheet using the following example of a data structure table:

Field Order	Data (Field)	Data Type	Max Length	Required	Unique
1	Item Number	Text	25	Yes	Yes
2	Serial Number	Text	25	No	Yes
3	Site	Text	25	Yes	No
4	Location	Text	25	Yes	No
5	Quantity	Integer	5	Yes	No

Table 3-1: Determine portable application's data storage requirements.

If you have determined that your storage structure should consist of multiple files, create an individual worksheet for each of your files; if necessary, make sure that you have defined a key field (please see Using Key fields on page 4-8 for more information).

Once you have begun building your UPG portable application, you can use the worksheet that you have developed as the blueprint for building your data files and the corresponding data fields.

Develop User Interface

The third and final phase of your project planning draws upon the information you have gathered in the previous two sections and helps you put together the user interface for your portable data collection application.

The first step in defining your user interface should be to determine the menus and sub-menus that your application will require. You can derive this information by reviewing your answers to the questions in the first part of the UPG Programming Methodology section; start with your portable application's main menu and work your way down. Your main menu may link directly to a tool or to a sub-menu. For example, your main menu may link to an inventory collection sequence (a tool) and a Utility menu (a sub-menu). Again, use a piece of graph paper to develop a flow chart for your menu structure. This will help you visualize your menu structure and also help you identify any potential problems. Be sure to include both links to sub-menus and tools.

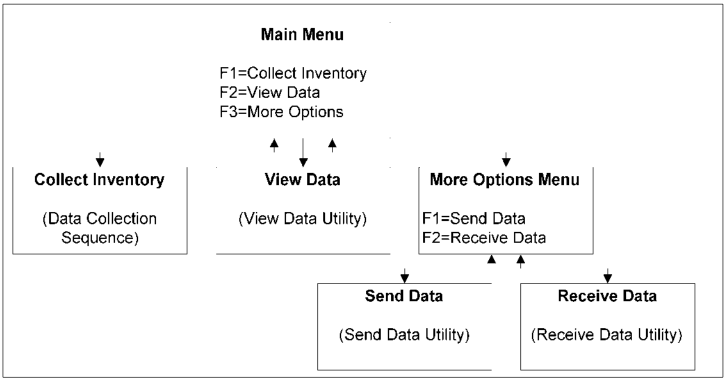


Figure 3-2: Sample flowchart of a data collection application.

Once you have specified your menu structure, it is time to define the actual tools called by the menus (recall that you may have tools intermixed with sub-menus). Return to your flowchart and identify the location of the tools that your menus call. The steps for determining which tools you need follows the same procedure as above; review your answers to the questions in the first part of the UPG Programming Methodology section. The information contained in these answers will help you determine which tools will satisfy the needs of your application. For example, if your application has a data collection requirement, you will need a multipage sequence tool to collect user input, or if you need to send collected data to a desktop PC, you will need a Send to PC tool that will upload a file from your portable unit to a host PC. To complete the process, specify on your flow chart the names and types of tools that your application requires.

Build Project Plan

When you have completed answering the previously stated questions, defined a clear application objective, and taken the time to develop a flow chart of your application, you essentially are holding the blueprint (or "Specs") for your portable application.

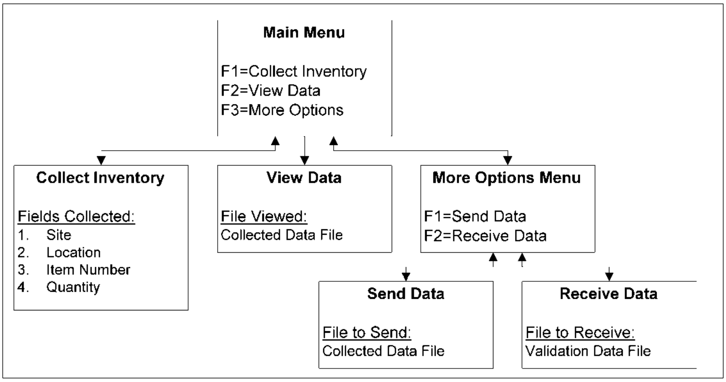


Figure 3-3: Sample flowchart of data collection application with specific details added.

You should now assemble all of the information that you have gathered and determine whether or not your planned application will meet your data collection requirements. When you have completed your review, begin to add details to your flowchart, such as fields that will be gathered in collection sequences and which files will be used in conjunction with what utilities (see Figure 3-3).

Use Project Plan

The steps you have just completed will help you to create an effective portable application in an efficient manner. When you begin to build your portable application, instead of haphazardly assembling an application, you will be able to logically construct an application that will meet the needs of you and your users based on the "blueprint" that you have just constructed.

When you sit down to build your portable application, the assembly of your portable application works basically in the same manner as building your project plan.

When you begin to build your portable application, you will want to create and name your application, build your storage structure, and then build your user interface.

- ❑ Use your Data Storage Requirement table (Table 3-1) to help you create your data files when using the File/Variable tooltype.
- ❑ Use your flowchart with added details (Figure 3-3) to build your user interface. This includes building menus, multipage sequences, and utilities.
    - Menus are created using the Form tooltype.
    - Data collection sequences are created using the Form tooltype (use the Multipage Sequence template).
    - Utilities are created using the Utility tooltype.

It is very important that you build your data storage structure before building your user interface simply because you need to point to your data structure from the user interface.

4

USING UPG

Introduction to UPG Programming

This chapter will teach you the fundamental skills needed to use the Universal Program Generator. To familiarize yourself with the skills needed, please complete the on-line tutorials. Select Tutorial from the Help menu and begin with the Before You Start selection. As you complete each tutorial level, continue with the next until you have completed at least the Level 2 tutorial.

Define Program

The Program tooltype is used to name your portable application and to specify which tool is automatically executed when you start your portable application.

1 **Double-click on the PROGRAM tooltype in the ToolBox.**
You can also right click on the PROGRAM tooltype in the TOOLBOX and select ADD.

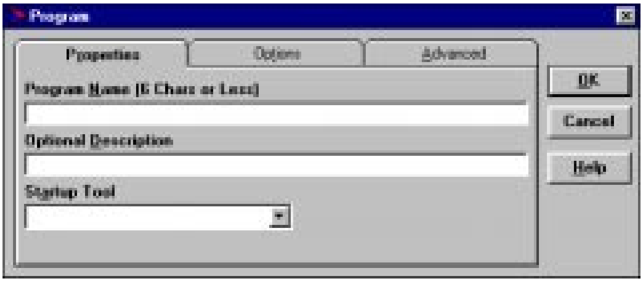


Figure 4-1: Program tooltype definition form.

2 **Enter the name of your application in the NAME field.**
The name of your application must be 6 characters or less to allow UPG to add a 2 character suffix to your application name when generating source code. For example, if your application name is `test.exe`, then the generated source code files will be named `test.c`, `test01.c`, `test02.c`, and so on.

Note: Do not add the .exe extension to your program name, UPG will do this automatically. When you press OK and return to the UPG work environment, your program tool will appear with the .exe extension (i.e. test.exe).

3 **Place a description for your program in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

4 **Specify the tool that will automatically be executed when your program is launched from the Startup Tool field (you have several options):**

Note: For most applications, your startup tool will be a “main menu.” This will allow your users to easily navigate between the different functions contained in your portable application. Menu tools are created using the Form tooltype; select Menu with (or without) Exit Key from the Select Template window (see Forms later in this chapter).

Programming from the ToolBox

If you choose to “program from the ToolBox,” you will first create all of the tools required for your UPG application(s) before actually creating a new UPG application (using the PROGRAM tooltype). Essentially, you will build an application in the following order:

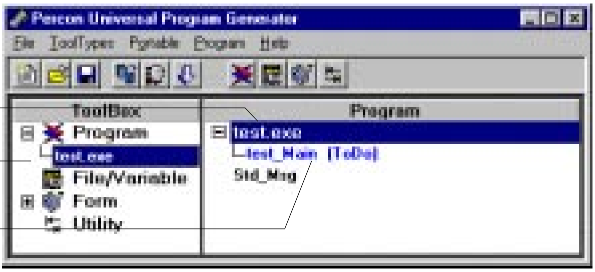
- □ Build File Structure
- □ Define Collection Sequences
- □ Build Utilities
- □ Build Menus
- □ Create Application (using PROGRAM tooltype)

The major advantage from building an application from the ToolBox is that it allows you to first build a collection of tools that will be shared between multiple applications; recall that it is possible for you to create more than 1 application (Program tool) per toolbox.

This approach allows you to maintain a certain consistency between similar products and, more importantly, will allow you to maintain/update multiple applications, while only having to maintain one set of tools.

After all of the required tools have been defined, constructing an application is merely connecting the different tools to one another. Connecting the tools is handled by both the Startup Tool field in the Program definition form, calling a tool from a hotkey, or using a UPG action component to call a tool.

a **Accept auto-generated tool**: When you enter the name of your application in the PROGRAM NAME field and press <TAB>, you will notice that a startup tool was created for you. The startup tool is derived from your program name and attaches "_main". For example, if your program name is `test`, your auto-generated startup tool will be `test_main`. If the auto-generated function does not exist in your toolbox, the startup tool will be labeled as a (To Do).



A tool created from the Program tooltype will appear in both the ToolBox and the Program pane.

Double-click on a (To Do) to convert it to a defined tool

Figure 4-2: A (To Do) denotes that a tool has been called (referenced) but is not currently defined (in the above example, test_main (To Do) was called from the Program Startup Tool field). This flexibility allows the developer to program in a "top-down" method.

b **Select an existing tool**: If you are working with a toolbox that already contains defined tools, you can select one of these to be your startup tool from the pull-down list.

c **Define your own tool**: Type your own tool name; if this tool is not currently defined in your toolbox, it will be labeled as a (To Do) in your Program pane.

Note: If you accept the auto-generated startup tool name or you define your own startup tool name and the tool does not exist in the open toolbox, the tool will be added to your program structure with the (To Do) label.

4 **Press OK when you finish defining your Program tool.** Your newly created tool will appear below the PROGRAM tooltype (this identifies that the tool was created with the Program tooltype) in your TOOLBOX. The tool will also appear at the top of your Program pane; note that the specified startup tool will also be attached below your `application_name.exe` in the PROGRAM pane.

Build Data Storage

Each of your portable data collection applications will require at least one data file; your data storage file will either be an ASCII or Fixed Length text file. If you will be validating collected data, you will also need to build validation files (see Chapter 5 for more information). More advanced applications, however, will also use memory variable and persistent INI files (each file type is explained in the following steps).

It is important to build your data storage structure before beginning any work on your collection sequences. If you attempt to build a collection sequence and the data storage does not exist, you will not be able to define any input (entry) fields in your multipage sequences (see the Forms section later in this chapter).

Please complete the following steps to build your data storage structure:

1 **Double-click on the FILE/VARIABLE tooltype in the ToolBox.** You can also right click on the FILE/VARIABLE tooltype in the TOOLBOX and select ADD.

2 **Highlight the FILE/VARIABLE template you want to use to create your file and press the OK button.**

- ❑ **ASCII**: Creates a file structure that is delimited (each field is separated by a delimiter). UPG automatically uses a comma as the field delimiter by default; press the OPTIONS tab to change the delimiter.
- ❑ **Fixed Length**: Creates a file structure that is based on the fixed length of each field; the fields are separated based on where one fixed length field ends and the next begins.
- ❑ **Memory Variable**: Creates a global C variable structure that is stored in your portable unit’s volatile memory (RAM); information stored in this file will be lost when your portable data collection application is terminated.
- ❑ **INI File (Persistent Variables)**: Creates a file structure similar to a memory variable file, but stores the information between application sessions (data is not lost when the application is terminated). This is useful for storing configu-

ration information that can be modified by the user of your portable data collection application.

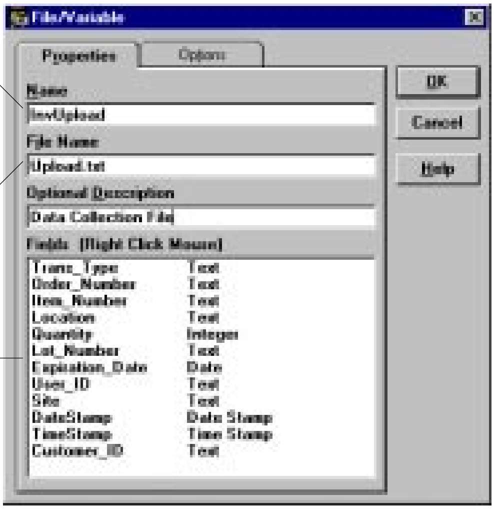
3 **Define the tool name for this file in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment (this is not the name of the DOS file that will be placed on your portable; see step 4). The DOS file name, however, will be derived from your tool name.

Note: Whenever you name a tool (using the NAME field), your entered value must comply with C programming naming conventions. Any invalid character will be converted to an underscore.

The value in the **Name** field is how you will refer to this file within the UPG development environment.

The value in the **File Name** field is the DOS name of your file.

The order of the fields in the **Fields** window determines the order that your collected data is written to the file.



Name: InvUpload

File Name: Upload.txt

Optional Description: Data Collection File

Fields (Right Click Mouse)

Trans_Type	Text
Order_Number	Text
Item_Number	Text
Location	Text
Quantity	Integer
Lot_Number	Text
Expiration_Date	Date
User_ID	Text
Site	Text
DateStamp	Date Stamp
TimeStamp	Time Stamp
Customer_ID	Text

Figure 4-3: File/Variable Definition Form

4 **Enter the DOS file name of your data file in the File Name field. Press <TAB>.**
You will notice that a DOS file name will automatically be created from the tool name entered in step 3. If you choose to leave the automatically generated DOS file name, skip this step.

If you would like to edit the DOS name, place your cursor in the File Name field and modify as necessary. The format of your file name must be `xxxxxxxx.txt` (an eight character or less name, followed by the .txt extension).

5 **Place a description for your file in the OPTIONAL DESCRIPTION field.**

This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

6 **Press the OPTIONS tab to configure the file's optional settings.**

Adjust your file's optional settings according to the following descriptions (for an explanation on using Key fields, please see the sidebar entitled, "Using Key Fields").

□ **Delimiter**: If you would like to specify a delimiter other than the default (a comma), enter the desired delimiter in this field. Be careful not to use characters that might be valid input (entry) values, such as an alpha character (a popular alternative to the comma is the semi-colon).

□ **Quotes**: Enable (check) this checkbox if you would like to surround your text fields with quotes

Note: Quotes are often specified when the data you are collecting includes descriptive text. For example, if you are using a comma as your delimiter and Quotes is enabled, text fields will be able to contain commas; if Quotes is disabled, then the text field cannot accept commas, because the field will be delimited prematurely. Also, quotes are sometimes required when interacting with a target application that uses the BASIC language.

□ **Create**: Enable (check) this checkbox if you want the currently selected data file to automatically be created on your portable if it does not already exist. This is an especially useful feature if the data file is deleted from the portable.

As a general rule of thumb, enable this checkbox on those files that are for data collection purposes. For files other than data collection, such as validation files, leave this fea-

USING KEY FIELDS

In very simple terms, defining a Key field establishes a sort pattern within the specified file. How does this improve searching performance? In technical terms, a key field will allow your application to perform a binary tree search, as opposed to a linear table search (please search on the word “key” in the online help for more information). In most cases, the Universal Program Generator will automatically create the proper keys for you.

1 **Verify that you are in a File/Variable definition form.**

2 **Press the Options tab.**

3 **Position your mouse in the Key window, right-click, and select Add Key.**

4 **Enter a name for your Key in the Name field.**

5 **If you would like to add a description for documentation purposes, place this description in the description field.**

6 **Highlight the field that you want to define as the "key" in the Field list and press the << button.**

7 **If you want to define another (secondary) key, repeat step 6.**

8 **Press the OK button.**

ture disabled. This will allow your portable to function properly if the validation files are missing.

Note: If you enable CREATE on a validation file, you will attempt to validate your data against an empty file, which will no doubt cause difficulties. See Chapter 5, Data Verification for more information on validation.

Add Fields to File

After you have defined the type of file you are creating, given it both a tool and DOS name, and configured optional settings, it is time to complete building your file by adding the data fields.

1 **Verify that you are currently in the Properties tab of the currently selected file.**
If you are in the Options tab, press the Properties tab to add data fields to your file.

2 **Position your cursor in the Fields window, right-click, and select Add Field.**
If you want to quickly add more than one field at a time, right-click in the Fields window, and select Batch Add.

3 **Enter the name of the field in the Field Name field.**

This is the field name that you will refer to while programming within the UPG environment.

4 **Select this field's type from the Field Type pull-down list.**

There are eight (8) types available for you to choose; please make your selection based on the following brief descriptions:

- ❑ **Text**: Will accept text input up to 128 characters in length (includes alpha and numeric characters, and punctuation characters).
- ❑ **Integer**: Accepts whole number (integer) input, up to 11 digits in length. It is declared as a type `long` in the UPG generated source code.
- ❑ **Floating Point**: Will accept decimal (real) numbers. It is declared as a type `double` in the UPG generated source code.
- ❑ **Date**: Accepts only valid dates. The length is automatically calculated and cannot be modified. Valid entry defaults to the mm/dd/yy format.
- ❑ **Time**: Accepts valid time input. The length is automatically calculated and cannot be modified. Valid entry defaults to hh:mm:ss.
- ❑ **Date Stamp**: Automatically writes the date that the record was collected to the file; this is not an input field and cannot be modified from within the UPG generated application.
- ❑ **Time Stamp**: Automatically writes the time that the record was collected to the file; this is not an input field and cannot be modified from within the UPG generated application.
- ❑ **Yes/No**: Accepts only an entry of Yes or No; press <Y> or <1> for an entry of Yes or <N> or <0> for a No entry. If the user enters a value of Yes, then a value of 1 is written to the file, whereas if a value of No is entered, than a value of 0 is written to the file.

5 **Specify the maximum field length in the Max Length field.**

This step only applies to Text, Integer, and Floating Point type fields; if you are not defining one of these fields, then you can skip this step.

6 **Place a description for your field in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

7 **Press the OK button when you finish defining your field.**
If you selected Batch Add in step 1, press the Cancel button on the blank form after you have completed defining your last field.

8 **When you complete defining all of the necessary fields for the currently selected file, press the OK button on the File/Variable window.**
The file you just finished building will appear below the File/Variable tooltype.

Note: Select Batch Add Fields in step 2 to repeatedly add fields to the currently selected file. When you press OK after defining the field, another field definition screen will automatically appear. Continue to define fields until you have finished adding all of the necessary fields to your file. Press the Cancel button to terminate the Batch Add process.

Create User Interface

After the program tool and data structures have been defined, you are now ready to create the user interface for your portable data collection application.

There are three major divisions of user interface components: menus, multipage sequences, and message boxes. All of these components are created using the Form tooltype. Please see the following sections for a detailed description on how to build your user interface using the Form tooltype.

Warning: If you do not have any data files defined, you will not be able to create any data collection sequences. Please see the Build Data Storage section earlier in this chapter.

Menus

Menus allow the users of your portable data collection application to navigate between the different functions that are contained within your application. Your application should contain at least a main menu that appears when you first start your application (your main menu will be the Startup Tool; see Define Program earlier in this chapter).

Typically, your main menu will contain a link to at least one collection sequence and several utilities. More advanced applications might have sub-menus that logically separate the different functions of your application.

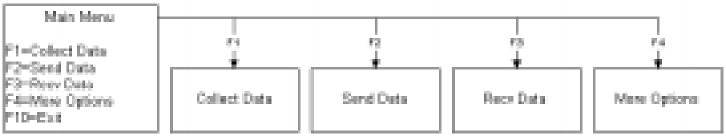


Figure 4-4: Sample flow chart of a main menu and its attached functions.

1 **Double-click on the Form tooltype in the ToolBox.**
You can also right click on the FORM tooltype in the TOOLBOX and select ADD.

Note: If you want to convert a (To Do) into a menu, double click on the desired (To Do) in the PROGRAM PANE and complete the following steps.

2 **Highlight the desired menu template and press the OK button.**

- ❑ **Menu with Exit Key**: A <F10> hotkey control will automatically be placed at the bottom of your menu. If the user of your portable application presses <F10> while in this form, the user will be returned to the previous menu. If this menu is the Startup Tool, then the application will be terminated and the user will be returned to a DOS prompt.

- ❑ **Menu without Exit Key**: No exit hotkey will be placed at the bottom of your menu. Your user, however, can press the <ESC> key to back up to the previous menu (<ESC> will not terminate the application).

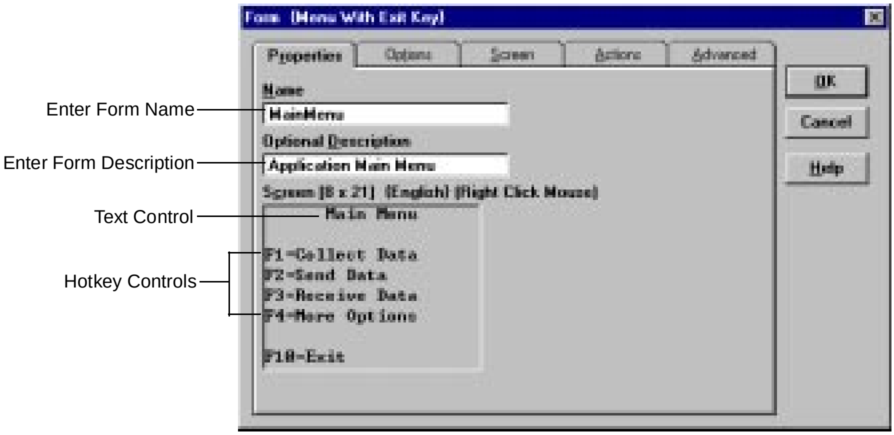
3 **Define the name for this form in the NAME field and press <TAB>.**

This is the name that you will refer to while programming within the UPG environment

If you are converting a (To Do) item into a menu, simply accept the default function name. This will ensure that any existing links (calls) remain intact.

4 **Place a description for your menu in the OPTIONAL DESCRIPTION field.**

This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.



Name: MainMenu

Optional Description: Application Main Menu

Screen [8 x 21] (English) (Right Click Mouse)

```
      Main Menu

F1=Collect Data
F2=Send Data
F3=Receive Data
F4=More Options

F10=Exit
```

Figure 4-5: Build your user interface with the individual controls in the Screen window; when your screen is complete, press the OK button to return to the UPG work environment.

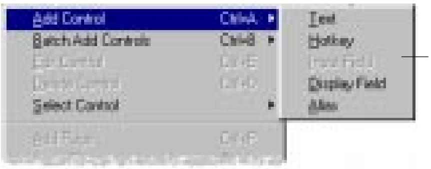
Add User Interface Controls: Text and Hotkeys

Controls are the items that allow the user to interact with your UPG generated portable application. Essentially, all of the items that are placed on your screen (text, hotkeys, input fields, display fields, and aliases) are controls.

Menus created with the Form tooltype can only use text, hotkey, and display field controls. The Text and Hotkey controls will be introduced during the following steps (the input field will be discussed in the Multipage Sequence section and the display field is discussed in the Message Box View File section later in this chapter):

1 **Verify that you are currently in the PROPERTIES tab of the currently selected form.**
If you are in a different tab, press the Properties tab to add user interface controls to your menu.

2 **Position your cursor in the SCREEN window, right-click, and select ADD CONTROL.**
If you want to quickly add more than one control of the same type at a time, right-click in the SCREEN window, and select BATCH ADD CONTROL.



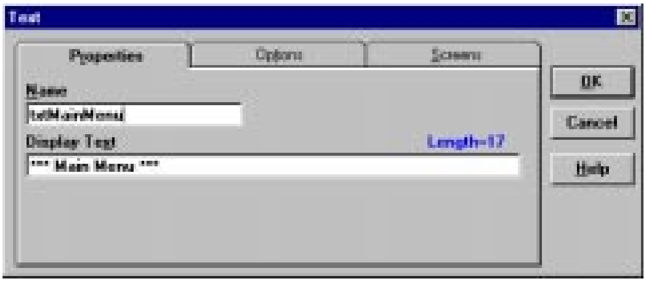
Available controls will be enabled, while unavailable controls, such as the Input Field control in this example, will be disabled.

Figure 4-6: As an alternative to right-clicking in the Screen window, you can also press <CTRL><A> and receive the pop-up menu at above right.

- □ **Text Control**: The text control allows you to place text labels on your form; this control is useful for titles and informative messages. To add a text control, complete the following:
  - a **Select Text from the pop-up menu (see Figure 4-6).**
  - b **Define the name of this text control in the NAME field and press <TAB>.**

This is the name that you will refer to while programming within the UPG environment.

Note: UPG automatically names each of your user interface controls; the name will be a composite of the control type and a numerical suffix. While your UPG application will function properly with these automatically generated names, your are strongly encouraged to give each of your controls meaningful names. This will help you identify the different controls while you are working with your application.



Name: txtMainMenu

Display Text: *** Main Menu ***

Length=17

Figure 4-7: Watch the Length counter above the Display Text field to make sure that your text label doesn't exceed the width of your portable screen.

c Enter the text for your control in the DISPLAY TEXT field and press <TAB>.

While entering your text, pay attention to the length of your text; there is a counter (LENGTH) above the display text field. Make sure that this value does not exceed the current width of your portables screen. If it does, the end of your display text will not be visible.

d Press the OK button when complete.

Your text control will appear in the screen window. The placement of your text control will default to the highest available position. If this is your first control, the text control will be placed at the top of the screen window.

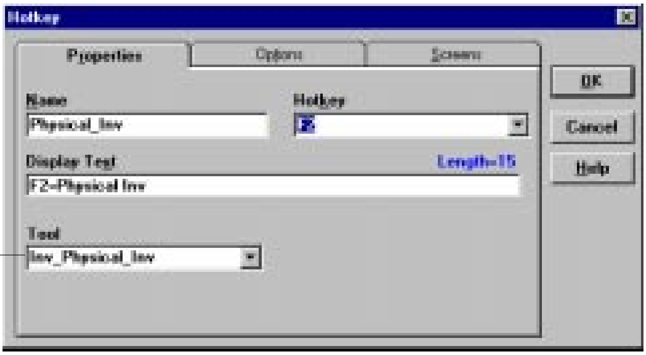
❑ **Hotkey Control**: A hotkey allows you to assign a tool (a form or utility) to an <F> key (or other available key combinations); this is an integral piece in your user interface.

a Define the name of this hotkey control in the NAME field and press <TAB>.

This is the name that you will refer to while programming within the UPG environment.

b Select the <F> key you want to use from the Hotkey pull-down list and press <TAB>.

In addition to the standard hotkeys (<F1> through <F10>), you can also select a variety of key combinations to function as a hotkey control (such as <CTRL><A>).



Name: Physical_Inv

Hotkey: F2

Display Text: F2-Physical Inv

Length=15

Tool: Inv_Physical_Inv

The automatically derived tool name will be a combination of your program name and the display text; you can select other tools from the pull-down list if desired.

Figure 4-8: Hotkey definition form.

c Enter the text for your control in the DISPLAY TEXT field and press <TAB>.

While entering your text, pay attention to the length of your text; there is a counter (LENGTH) above the display text field. Make sure that this value does not exceed the current width of your portable screen. If it does, the end of your display text will not be visible.

d In the TOOL OR ACTION field, specify the tool that will be executed when this hotkey is pressed (you have several options):

Accept auto-generated tool: When you enter the display text and press <TAB>, you will notice that a tool is automatically generated from the display text. For example, if your program name is `test` and you entered `Collect Data` in the DISPLAY TEXT field, your auto-generated tool will be `test_Collect_Data`. If the auto-generated tool does not exist in your toolbox, the tool will be placed in the program pane and labeled as a (To Do).

Select an existing tool: If you are working with a toolbox that already contains defined tools, you can select one of these to be the executed tool from the pull-down list.

Define your own tool: Type your own tool name; if this tool is not currently defined in your toolbox, it will be labeled as a (To Do) in your Program pane.

Note: If you want to execute a UPG Action component using a hotkey, enable (check) the CALL FORM ACTION checkbox (this checkbox is only available when you have created UPG Action components for this form - see Chapter 8 for more information). A list of all Action components defined in this form will be available in the TOOL OR ACTION pull-down list; select the Action component that you want to execute when the user presses the currently selected hotkey.

5 **Press the OK button when you have finished defining your control.**
If you selected Batch Add in step 1, press the CANCEL button on the blank form after you have completed defining your last control.

You have just completed adding a control to your form. To add another control to your form, repeat the Add User Interface Controls section.

When you have completed defining your form, press the OK button on the Form definition screen to return to the UPG work environment.

Multipage Sequences

The multipage sequence is the Form template that enables you to build the collection sequences. A multipage sequence allows you to retrieve data input from the users of your portable data collection application. A unique difference between the multipage sequence and menu or message box form is that the multipage sequence may have multiple pages. In other words, a single multipage sequence tool may contain several pages (or screens) worth of information.

This flexibility allows you to build collection sequences that contain more than 2 or 3 input fields (any more than 2 or 3 input fields on a single page becomes quite cluttered, and is difficult to use).

In addition to the text and hotkey controls introduced in the Menus section, multipage sequences can also use the input field control. In fact, the input field control can only be used in conjunction with the multipage sequence. This is the actual control that receives the data input from your users. Please complete the following steps to build a multipage sequence form.

1 **Double-click on the Form tooltype in the ToolBox.**
You can also right click on the FORM tooltype in the TOOLBOX and select ADD.

Note: If you want to convert a (To Do) into a multipage sequence, double click on the desired (To Do) in the PROGRAM PANE and complete the following steps.

2 **Highlight the multipage sequence template and press the OK button.**

3 **Define the name for this form in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment

If you are converting a (To Do) item into a multipage sequence, simply accept the default function name. Accepting will ensure that any existing links (calls) remain intact.

4 **Place a description for your multipage sequence in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

Select the file that you want to add the collected data to.

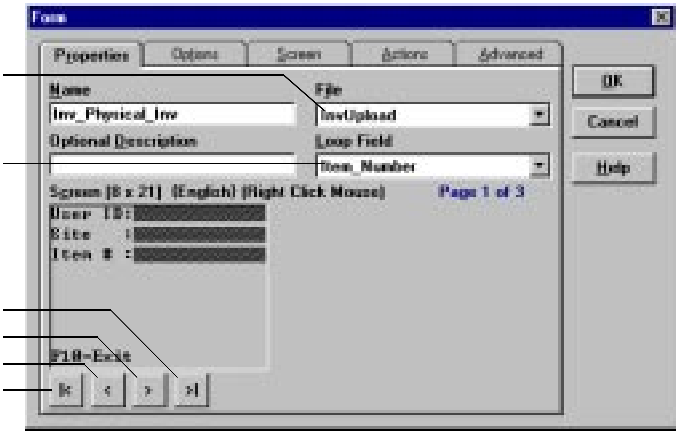
Select the input field to loop back to when the end of the multipage sequence is reached.

Last Page

Next Page

Previous Page

First Page



Name: Inv_Physical_Inv

File: InvUpload

Optional Description:

Loop Field: Item_Number

Screen [8 x 21] (English) (Right Click Mouse) Page 1 of 3

```
User ID:
Site   :
Item # :



F10=Exit
```

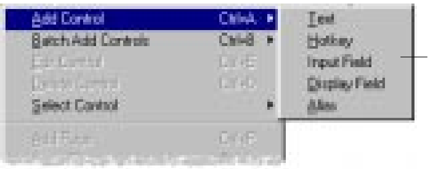
Figure 4-9: Notice the similarities and differences between a form definitions screen. Compare the above multipage sequence template and a menu template (see **Figure 4-5**).

Add User Interface Controls: Input Fields

If you recall from the previous Menu section, controls are the items that allow the user to interact with your UPG generated portable application. Since the Text and Hotkey controls were introduced in the Menus section, only the input field control will be introduced in this section (the display field is discussed the in Message Box View File section later in this chapter):

1. **Verify that you are currently in the PROPERTIES tab of the currently selected form.**
   If you are in a different tab, please press the Properties tab to add user interface controls to your menu.

2. **Position your cursor in the SCREEN window, right-click, and select ADD CONTROL.**
   If you want to quickly add more than one control of the same type at a time, right-click in the SCREEN window, and select BATCH ADD CONTROL. When you use Batch Add Control, a new page will be created for each new control that you create. This is a very quick method to build a collection sequence, but it does not allow you to place multiple input fields on a single page.



Notice that all control types are available when working with a multipage sequence form. The input field control receives user input.

Figure 4-10: As an alternative to right-clicking in the Screen window, you can also press <CTRL><A> and receive the pop-up menu at above right.

❑ **Input Field Control**: The input field control receives data input from the user. Each input field is linked to the fields that are defined in your File/Variable tools. These links determine where the data entered by the user is saved. To add an input field control, complete the following:

Note: If you currently do not have any File/Variable tools specified, please see the Build Data Storage section earlier in this chapter. If no File/Variable tool is currently defined, you will not be able to add input field controls to your multipage sequence.

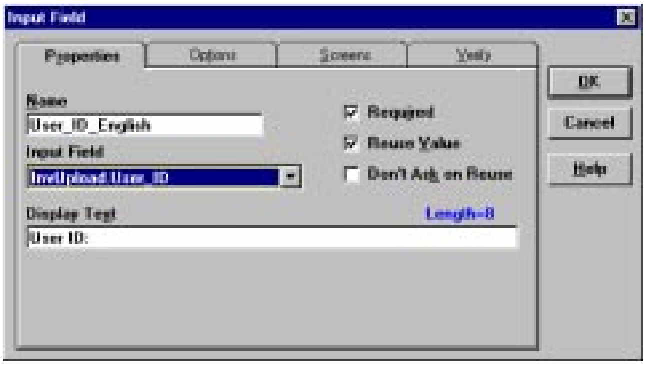
a **Select Input Field from the pop-up menu (see Figure 4-10).**

b **Define the name of this input field control in the NAME field and press <TAB>.**

This is the name that you will refer to while programming within the UPG environment. It is especially important to name your input fields with meaningful names. You will find that you will refer to these fields often, especially when using UPG Actions (see Chapter 8, UPG Actions).

c **Select the target field in which you will store the collected data from the INPUT FIELD pull-down list (see Figure 4-11).**

The target fields are listed as the composite of the file/variable name and the field name. For example, if you have a File/Variable tool named `CollectData` and a field that is named `ItemNo`, the field will be listed in the pull-down list as `CollectData.ItemNo`. This convention allows you to identify the proper target field (this is especially handy when you have multiple files with similar field names).



Name: User_ID_English

Input Field: InvUpload.User_ID

Display Text: User ID:

Length=8

☑ Required

☑ Reuse Value

□ Don't Ask on Reuse

Figure 4-11: Input field definition form.

d **Enter the text for your control in the DISPLAY TEXT field and press <TAB>.**

Pay attention to the length of your text; there is a counter (LENGTH) above the display text field. Make sure that this value does not exceed the current width of your portables screen. If it does, the end of your display text will not be visible.

e **Modify the input field property settings.**

You can configure several property settings for each input field; you can turn the settings on by enabling the appropriate checkbox. Please adjust your settings according to the following descriptions:

Required: If enabled, the user will be required to enter a value into this input field. If the user attempts to advance to the next input field without entering a value in the current input field, the user will receive a message stating that an entry is required. If this feature is disabled, the user will be able to leave the field blank.

Reuse Value: If enabled, the value entered into this field during the first pass of the collection sequence will automatically be placed in this field during subsequent passes through the collection sequence. The user can press <ENTER> or <TAB> to accept this value or modify the value as desired. This feature is useful for collecting data that will often stay the same, such as locations when collecting inventory data. If disabled, the input field will always default to a blank value.

Don’t Ask on Reuse: This feature is only available if you have enabled Reuse Value. Enable this checkbox to automatically accept the reused value after the first pass through the collection sequence. The user will not be able to modify the value, and in essence, this field will be skipped after the first pass. Therefore, the value entered during the first pass will be used for all subsequent passes.

3 **Press the OK button when you have finished defining your Input Field control.**
If you selected Batch Add in step 1, press the CANCEL button on the blank form after you have completed defining your last control.

Continue to add input fields (or other controls) to this form. Remember, it is a good idea to limit your input fields to 2 or 3 per page.

Add Another Page

As the name implies, a multipage sequence can contain more than one page (or screen) in the collection sequence. After the user enters a value into the last field on a particular page, the user will automatically be advanced to the following page. Multipage sequences are especially useful if you are working with a portable that has a small display. Instead of “cramming” multiple fields onto a small screen, you can spread the required input fields between multiple pages.

Complete the following steps to add another page to your multipage sequence:

1 **Position your cursor in the Screen window, right-click, and select Add Page.**
As a shortcut, you can also press <CTRL><P> to add another page to your multipage sequence.

2 **Continue to add controls as described in the previous sections.**
Adding controls to a new page requires the same steps as previously described. Think of the added pages as a mere extension of the first page in your collection sequence.

Tip: If you want to insert a page between two currently defined pages, open the page after the desired insertion point, right-click in the Screen window, and select Insert Page from the pop-up menu.

Complete the Multipage Sequence

There are several more steps that you need to complete before you can consider your multipage sequence finished. You need to specify which file you will be saving your data to and which field you want to “loop” on (this pull-down list will determine where each subsequent pass through the collection sequence starts). Complete the following steps to finish creating your multipage sequence:

1 **Select the file that you want to add the collected data to from the FILE pull-down list.**
After the user enters a value into the last input field contained in your multipage sequence, the collected data will be added as a complete record to the file specified in the FILE pull-down list.

2 **Select the input field that you want to return to after a value has been entered in the last input field of the current multipage sequence.**
To expedite data collection, you want to make it easy for your users to collect repetitive data. Using a loop field can allow users to collect data rapidly (because they can repeatedly collect data without ever leaving the multipage sequence). You can choose any input field in the current multipage sequence as your loop field. Usually, you may return to the second or third input field in your collection sequence, since the first couple of input fields may contain information that stays constant for a data collection routine, such as user or site information.

3 **Press the OK button to return to the UPG work environment when you have completed building your multipage sequence.**

You will notice that when you return to the UPG work environment that the file(s) referenced in your multipage sequence will appear attached to your multipage sequence tool.

Message Boxes

As the name implies, the message box Form template allows you to create informative screens for your portable application. This template serves as a mechanism to communicate information to the users of your portable data collection application. This form template, however, does not allow user interaction (you can only use Text and Display field user interface controls with this form template).

1 **Double-click on the Form tooltype in the ToolBox.**

You can also right click on the FORM tooltype in the TOOLBOX and select ADD.

Note: If you want to convert a (To Do) into a message box, double click on the desired (To Do) in the PROGRAM PANE and complete the following steps.

2 **Highlight the desired message box template and press the OK button.**

- □ **Message Box with Any Key Message**: An “Any key to continue” message will automatically be placed at the bottom of your message box. The user can dismiss the message box by simply pressing any key.
- □ **Message Box without Any Key Message**: No message will appear at the bottom of your message box, though the user can still dismiss the message box by pressing any key.

3 **Define the name for this form in the NAME field and press <TAB>.**

This is the name that you will refer to while programming within the UPG environment

If you are converting a (To Do) item into a menu, simply accept the default name; this will ensure any existing links (calls) will remain intact.

4 **Place a description for your menu in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

Add User Interface Controls

Only text and display field controls can be used with a message box form. Text controls were introduced in the Add User Interface Controls: Text and Hotkeys section and the display field control will be introduced in the Message Box View File section below.

1 **Position your cursor in the Screen window, right-click, and select the desired control.**
Select the text control to add a text label (or title) to your message box or select the display field control to display the result of a field, calculation, or variable.

2 **Press the OK button when you have completed creating your control.**

3 **Repeat steps 1 and 2 until you have completed entering all of the required controls for your message box.**
If you are adding a long text message to your message box, you may need to use several text controls to display your entire message (the text control does not handle text wrapping).

4 **Press the OK button when you have finished creating your message box.**

Message Box View File

The Message Box View File template is to be used in conjunction with the View/Edit File utility. It is a custom message box that contains record navigation labels at the bottom of the form. To view the data, you will use the Display Field control to display your data in the format you want. Using a Message Box View File template to create a form to view data will enable you to view your data in a meaningful way.

1 **Double-click on the Form tooltype in the ToolBox.**
You can also right click on the FORM tooltype in the TOOLBOX and select ADD.

Note: If you want to convert a (To Do) into a message box, double click on the desired (To Do) in the PROGRAM PANE and complete the following steps.

2 **Highlight the Message Box View File template and press the OK button.**

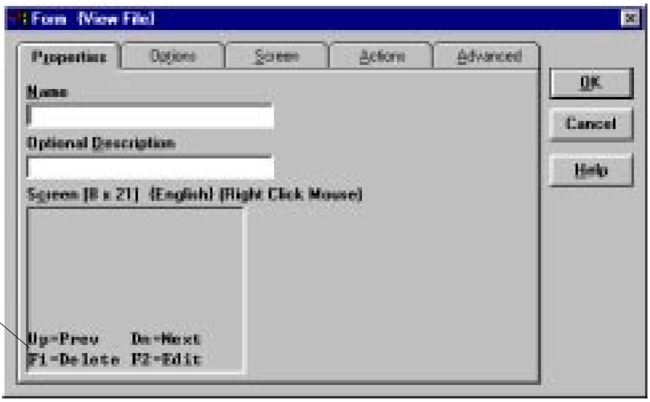
3 **Define the name for this form in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment

If you are converting a (To Do) item into a menu, simply accept the default function name; this will ensure that any existing links (calls) remain intact.

4 **Place a description for your menu in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

Add User Interface Controls: Display Field

Only text and display field controls can be used with a message box form. Text controls were introduced in the Add User Interface Controls: Text and Hotkeys section. The Display Field control will be introduced in this section.



```
Up=Prev    Dn=Next
F1=Delete  F2=Edit
```

Record navigation controls are automatically listed at the bottom of a Message Box View File template; these remind the user how to view the collected data.

Figure 4-12: The same controls that are available with one of the Message Box templates are available with the Message Box View File template; use Display Field controls to view your collected data.

5 **Position your cursor in the Screen window, right-click, and select Display Field.**

You can also add text controls to a tool created with the Message Box View File template, but since display field controls have a display text field, you won't have much cause to use the text tool (remember, portable screen space is at a premium and you only have one screen to display your data).

6 **Define the name of this display field control in the NAME field and press <TAB>.**

This is the name that you will refer to while programming within the UPG environment.

7 **Select the field that contains the data you want to display from the DISPLAY FIELD pull-down list.**

The fields are listed as the composite of the file/variable name and the field name. For example, if you have a File/Variable tool named `CollectData` and a field that is named `ItemNo`, the field will be listed in the pull-down list as `CollectData.ItemNo`. This convention allows you to identify the proper field (this is especially handy when you have multiple files with similar field names).

8 **Enter the text for your control in the DISPLAY TEXT field and press <TAB>.**
The display text will default to the name of the field. If this is not suitable, you may edit the display text. Pay attention to the length of your text; there is a counter (LENGTH) above the display text field. Make sure that this value does not exceed the current width of your portables screen. If it does, the end of your display text will not be visible.

9 **Press the OK button when you have completed creating your display field.**

10 **Repeat steps 5 and 9 until you have completed defining all of the required display fields.**

11 **Press the OK button when you have finished to return to the UPG work environment.**

Add Utilities

The Utilities tooltype enables you to define a variety of utilities to be used in your UPG generated application. As of this release, there are seven available Utility templates; these templates typically allow you to build communication and file utilities, while others allow the user to exit to DOS or execute a DOS command.

Complete the following steps to add one of the available utilities to your UPG generated application:

1 **Double-click on the Utility tooltype in the ToolBox.**
You can also right click on the UTILITY tooltype in the TOOLBOX and select ADD.

Note: If you want to convert a (To Do) into a Utility tool, double click on the desired (To Do) in the PROGRAM PANE and complete the following steps.

2 **Highlight the desired menu template and press the OK button.**

- □ **Delete File**: Allows the user to delete a specified file (useful when the user want to clear or reset the data collection file after sending the data to a target application).

- ❑ **Exit to DOS**: This utility closes the UPG generated application and returns the user to a DOS prompt.
- ❑ **File Transfer Host Mode**: Places your portable in the transfer host mode (will continue to receive files until you end the transfer session).
- ❑ **Receive File from Host**: Use this utility when you need to transfer a data file from a host PC to your portable data collection unit.
- ❑ **Run External DOS Program**: Allows the user to define an executable command to launch another DOS program on the portable data collection unit.
- ❑ **Send File to Host**: This utility allows you to transfer a data file from your portable data collection unit to a target PC.
- ❑ **View/Edit File**: The View/Edit File utility enables you to view the contents of a file from within your portable data collection application.

3 **Complete the steps contained in one of the following sections that matches the utility template you have chosen in step 2.**

Delete File

The Delete File utility allows you to define a tool that will delete a specified file on your portable data collection unit.

1 **Define the name for this utility in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

If you are converting a (To Do) item into a Delete File utility, simply accept the default tool name. This will ensure that any existing links (calls) remain intact.

2 **Place a description for your Delete File utility in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

3 **Select the file that you want to delete from the File pull-down list.**
This pull-down list will list all currently defined file/variable tools. Excluded from this list are any file/variable tools that were created with the Memory Variable template.

4 **Press the OK button when you have completed defining your Delete File utility.**

Exit to DOS

The Exit to DOS utility allows you to define a tool that will terminate your portable data collection application and return you to the DOS environment.

1 **Define the name for this utility in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

If you are converting a (To Do) item into an Exit to DOS utility, simply accept the default tool name. Accepting the name will ensure that any existing links (calls) remain intact.

2 **Place a description for your Exit to DOS utility in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

3 **Press the OK button when you have completed defining your Exit to DOS utility.**

File Transfer Host Mode

The File Transfer Host Mode utility allows you to define a tool that places your portable in the host mode. Host mode means that your portable will receive any file that is sent to it, and will continue to receive files until the user press <ESC> to cancel the host mode. This utility will very helpful when you want to send a series of files to the portable with one transfer session (this is advantageous over using the Receive File from Host, because you can only receive one file at a time).

1 **Define the name for this utility in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

If you are converting a (To Do) item into a File Transfer Host Mode utility, simply accept the default tool name; this will ensure that any existing links (calls) remain intact.

2 **Place a description for your File Transfer Host Mode utility in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

3 **Press the OPTIONS tab.**

4 **If you would like to give your users the ability to receive this file via a modem connection and/or direct connection, enable the LET USER SPECIFY DESTINATION checkbox.**
When the user activates this feature in the UPG generated application, they will be presented with the question: Use Modem? The UPG generated application will default to NO.

If the user accepts the default of No, then a direct connection will be used. If the user specifies Yes, then the UPG generated application will receive the file via modem.

Both communication options will use the appropriate default settings specified in the COMMUNICATIONS SETUP (available from the PORTABLE menu)

If you want to limit the user to only direct connects, disable the LET USER SPECIFY DESTINATION checkbox. If this checkbox is disabled, then when the user activates this feature, the portable application will automatically use a direct connection.

5 **Press the OK button when you have completed defining your File Transfer Host Mode utility.**

Note: See Chapter 10, UPG Communications for more information on using and setting up portable communications.

Receive File from Host

The Receive File From Host utility allows you to define a tool that will allow your portable data collection application to receive a file from the host PC. This may be very helpful when you want to transmit validation data from the host PC to your portable unit.

1 **Define the name for this utility in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

If you are converting a (To Do) item into a Receive File from Host utility, simply accept the default tool name. This will ensure that any existing links (calls) remain intact.

2 **Place a description for your Receive File from Host utility in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

3 **Select the file that you want to receive from the host PC from the FILE pull-down list.**
This pull-down list will list all currently defined file/variable tools. Excluded from this list are any file/variable tools that were created with the Memory Variable template.

4 **Press the OPTIONS tab.**

5 **If you would like to give your users the ability to receive this file via a modem connection and/or direct connection, enable the LET USER SPECIFY DESTINATION checkbox.**
When the user activates this feature in the UPG generated application, they will be presented with the question: Use Modem? The UPG generated application will default to NO.

If the user accepts the default of No, then a direct connection will be used. If the user specifies Yes, then the UPG generated application will receive the file via modem.

Both communication options will use the appropriate default settings specified in the COMMUNICATIONS SETUP (available from the PORTABLE menu)

If you want to limit the user to only direct connects, disable the LET USER SPECIFY DESTINATION checkbox. If this checkbox is disabled, then when the user activates this feature, the portable application will automatically use a direct connection.

6 **Press the OK button when you have completed defining your Receive File from Host utility.**

Note: See Chapter 10, UPG Communications for more information on using and setting up portable communications.

Run External DOS Program

The Run External DOS Program utility allows you to define a tool that will allow your portable data collection application to run a DOS application stored on your portable data collection unit.

1 **Define the name for this utility in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

If you are converting a (To Do) item into a Run External DOS Program utility, simply accept the default tool name. This will ensure that any existing links (calls) remain intact.

2 **Place a description for your Run External DOS Program utility in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

3 **Enter the command line required to execute your external DOS program.**
Make sure that you specify any required command line parameters and to include a fully qualified path for your executable statement (i.e. - `c:\itrak\inv`).

4 **Press the OK button when you have completed defining your Run External DOS Program utility.**

Send File to Host

The Send File From Host utility allows you to define a tool that will allow your portable data collection application to send a file to the host PC; this utility allows you to transfer the data you have collected with your portable data collection application to a host PC.

1 **Define the name for this utility in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

If you are converting a (To Do) item into a Send File to Host utility, simply accept the default tool name. This will ensure that any existing links (calls) remain intact.

2 **Place a description for your Send File to Host utility in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

3 **Select the file that you want to send to the host PC from the File pull-down list.**
This pull-down list will list all currently defined file/variable tools. Excluded from this list are any file/variable tools that were created with the Memory Variable template.

4 **Press the OPTIONS tab.**

5 **If you would like to give your users the ability to send this file via a modem connection and/or direct connection, enable the LET USER SPECIFY DESTINATION checkbox.** When the user activates this feature in the UPG generated application, they will be presented with the question: Use Modem? The UPG generated application will default to NO.

If the user accepts the default of No, then a direct connection will be used.

If the user specifies Yes, then the UPG generated application will send the file via modem. Also, after the user specifies yes to use the modem, the user will have the option to change the telephone number or accept the default phone number; any change will be saved for the next session.

Both communication options will use the appropriate default settings specified in the COMMUNICATIONS SETUP (available from the PORTABLE menu)

If you want to limit the user to only direct connections, disable the LET USER SPECIFY DESTINATION checkbox. If this checkbox is disabled, then when the user activates this feature, the portable application will automatically use a direct connection.

6 **Press the OK button when you have completed defining your Send File to Host utility.**

Note: See Chapter 10, UPG Communications for more information on using and setting up portable communications.

View/Edit File

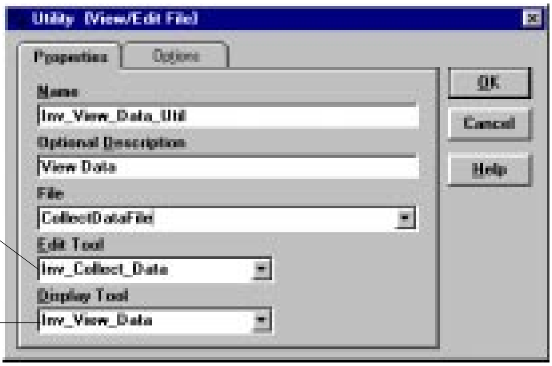
Using the View/Edit File utility will allow the users of your portable data collection application to view and/or edit data in the specified file. You have the option of allowing the user to view the data in a raw text format, or you can use form tools to display and edit your data (using a form tool will be more intuitive than looking at a raw text file).

1 **Define the name for this utility in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

If you are converting a (To Do) item into a View/Edit File utility, simply accept the default function name. This will ensure that any existing links (calls) remain intact.

2 **Place a description for your View/Edit File utility in the OPTIONAL DESCRIPTION field.**
This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

3 **Select the file that you want to view and/or edit from the File pull-down list.**
This pull-down list will list all currently defined file/variable tools; excluded from this list are any file/variable tools that were created with the Memory Variable template.



Name: Inv_View_Data_Util
Optional Description: View Data
File: CollectDataFile
Edit Tool: Inv_Collect_Data
Display Tool: Inv_View_Data

Select a *form* tool made with the Multipage Sequence template.

Select a *form* tool made with the Message Box View

Figure 4-13: If you don’t select an edit or display tool, you will edit and view your data in the raw ASCII format.

4 **Select the editing tool you want to use from the EDIT TOOL pull-down list.**

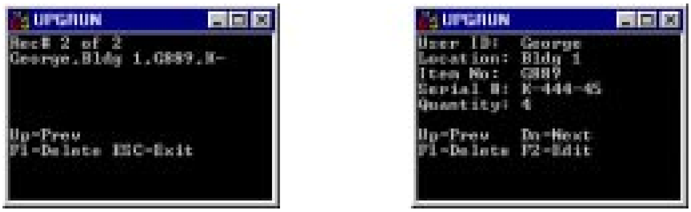
Select a multipage sequence to edit your data. You will most often want to select the same multipage sequence that is used to collect the data for this file.

When the user activates this utility, the user only needs to press the <F2> button to enter the edit mode.

5 **Select the viewing tool you want to use from the DISPLAY TOOL pull-down list.**

Select a tool that was created with the Message Box View File form template (please see the Message Box View File section earlier in this chapter).

If a tool is not selected, the user will only be able to view the data in its raw text format (see the comparison in **Figure 4-14**). Depending on the file type, the fields will be separated by a delimiter or determined by the fixed length of each field (see Build Data Storage earlier in this chapter for more information).



```
Rec# 2 of 2
George.Bldg 1.G889.K-



Up-Prev
F1-Delete ESC-Exit
```

```
User ID:   George
Location:  Bldg 1
Item No:   G889
Serial #:  K-444-45
Quantity:  4

Up-Prev    Dn-Next
F1-Delete  F2-Edit
```

Figure 4-14: Note the differences between specifying a view tool and not specifying a view tool. The screen at above left does not have a view tool specified and allows you to view the data in the raw ASCII format only. The screen at above right has a view tool specified. It was built using the Message Box View File template and Display Field controls; it presents the data in a more readable format.

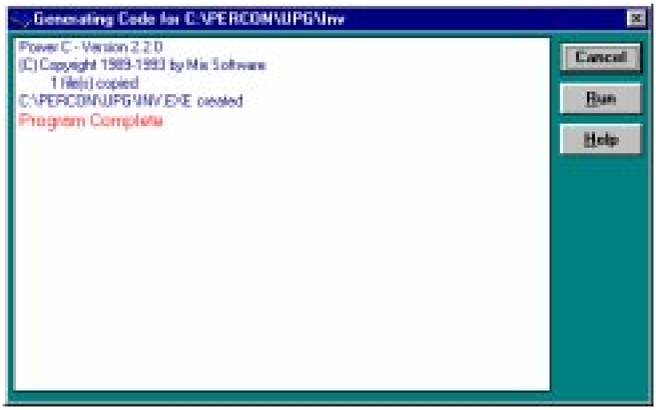
6 **Press the OK button when you have completed defining your View/Edit File utility.**

Compile/Test Project

After you have completed putting together your portable data collection application, it is time to compile and test your application. UPG has automated the compiling process. Simply complete the following steps to compile your application.

1 **Make sure that the UPG security key is properly fastened to your LPT1 (printer) port.**
You will not be able to compile your portable data collection application unless the UPG security key is attached to your computer.

2 **Select BUILD PROGRAM from the PROGRAM menu.**
BUILD PROGRAM will generate the C source code and compile your UPG data collection application. You can also press <F5> to build your program.



```
Power C - Version 2.2.0
(C) Copyright 1989-1993 by Mix Software
      1 file(s) copied
C:\PERCON\UPG\INV.EXE created
Program Complete
```

Figure 4-15: The message “Program Complete” will appear in the Compile window when your application has been compiled. Press the Run button to execute your portable application in a DOS “box.”

3 **When your application has been compiled, press the Run button to launch your portable application.**
Your portable application will be run in a Windows DOS “box.” You can run your application exactly as it will run on your DOS based portable data collection unit.

It is much easier to test/debug a portable application on your development PC rather than having to re-program your portable each time. When your portable application has become stable and meets your requirements, then you should test your portable data collection application on your portable.

Note: If you receive compile errors, press the HELP button on the Compiler Error screen to help you debug your portable application.

Program Portable

When you have completed building, compiling, and testing your portable application, it is time to program your portable. However, since UPG supports several different portable platforms, all matters of communication including programming your portable are covered in Chapter 10, UPG Communications.

5 DATA VERIFICATION

UPG gives you plenty of tools to make sure that the data being entered into your portable data collection application is valid data. Data verification can greatly increase the accuracy of the data being entered in your data collection applications, and in turn, can increase the quality of your data collection operations. Consequently, an increase in quality can reduce the costs of correcting data collection errors.

Validation

Validation compares the value being entered into your portable data collection application with a value contained in a validation file; if the values do not match, then the user will receives a message stating that the entered value is not valid. As the developer, you can give the user the ability to override such a message, or you can force a valid entry.

Validation Data Structure

Before you begin to apply validation to your input fields, you need to make sure that you have the file structure defined for validation files that you will be using. If your validation files are not defined, then you will not be able to identify what file/field your entered values will be compared against.

Building a validation file entails the same exact steps as defining a data collection file, with one exception: make sure that you do not enable the Create checkbox in the Options tab of the File/Variable definition screen. If you were to choose Create and the validation file did not exist, your UPG generated application will create a blank validation file, and all of your entered data would obviously not be valid.

Tip: When building your validation files, it is sometimes helpful to use the same field names that are specified in your data collection application (you will not be able use the same file name). This will make identifying the proper Lookup Field for validation much easier.

Please see Chapter 4, Using UPG for more information. The Build Data Storage section provides all of the necessary instructions for building a file structure.

Apply Validation

Validation is defined at the input field level. In the previous chapter, you learned how to define an input field when building a multipage sequence.

1 **Open the multipage sequence that will contain validation.**

2 **Make sure that you have the input field definition screen open for the input field you want to validate.**
You can add validation when you first add an input field to a multipage sequence, or you can add validation to an existing input field.

If you want to add validation to an existing input field, complete the following to edit an input field:

a **Position your cursor in the Screen window, right-click, and select Edit Control.**

b **Select the input field that you want to edit from the pop-up menu.**

3 **Press the Verify tab to access the validation functions.**
All data verification functions are contained within the Verify tab of an input field definition screen.

4 **Complete the individual steps contained in the following two sections: Define Lookup Validation and Configure Validation Options.**
The following sections will describe in detail how to define validation functions for the selected input field.

Define Lookup Validation

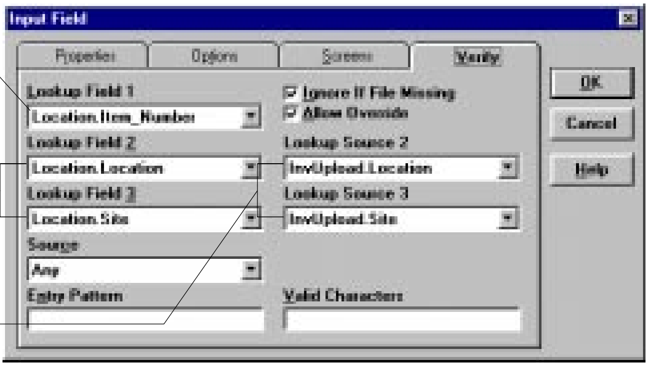
The lookup field specifies what the entered value (lookup source) is compared against (lookup field). For example, if you entered an ItemNo value in the `collect.txt` file, you might select ItemNo in the `validate.txt` file as the lookup field.

You can easily specify what the lookup source is compared against by selecting the appropriate file/field from the appropriate Lookup Field pull-down lists.

1 **Select the field you want to validate against from the Lookup Field 1 pull-down list.**
As elsewhere, the Lookup Field pull-down list will display available fields in the file.field format. The value that is entered into the currently selected input field will be validated (or compared) against the field that is selected in the Lookup Field 1 pull-down list. If a match is found within the file, then the entered value will be valid, and conversely, if a match is not found, the user will receive a warning that the entered value is not valid.

Note: The lookup source for the Lookup Field 1 is the currently selected input field.

2 **Define hierarchical validation.**
UPG can handle three (3) levels of validation (multiple levels of validation is typically referred to as hierarchical validation). A valid entry will require that all data entered into the currently selected input field and those fields selected in the Lookup Source contain valid data.



Lookup Field 1: Location.Item_Number
☑ Ignore If File Missing
☑ Allow Override
Lookup Field 2: Location.Location
Lookup Source 2: InvUpload.Location
Lookup Field 3: Location.Site
Lookup Source 3: InvUpload.Site
Source: Any
Entry Pattern:
Valid Characters:

Compares the value entered into the currently selected input field against the selected `file.field`.

Lookup Fields 2 and 3 validates the `file.field` selected in the associated Lookup Source pull-down

Specifies which input fields will be validated by the associated Lookup Field pull-down list.

Figure 5-1: Verify tab of an Input Field definition screen. The fields selected in the Lookup Source pull-down lists should precede the currently selected input field in the collection sequence; if they do not, then your portable application will attempt to validate null (blank) values.

a **Select the file.field from the LOOKUP FIELD 2 pull-down list that you want to compare the lookup source against.**

The value that is entered in the Lookup Source 2 field will be compared against the file.field that is specified in the Lookup Field 2 pull-down list.

b **Select the file.field that you want validated from the LOOKUP SOURCE 2 pull-down list.**

Logically, the value that is entered into this field will be validated against the field that is specified in the Lookup Field 2 pull-down list.

Note: When using a hierarchical validation for a particular input field, the value entered into the input field will not be a valid entry unless all specified fields in the hierarchy are valid. A common use would be to not only validate an Item#, but also the Site and Location. If the Item# and Location contained valid entries, but the Site did not, the Item# would not be a valid entry.

c **If a third level of validation is needed, repeat steps a and b above, but use the Lookup Field 3 and Lookup Source 3 pull-down lists.**

If you want to accept the default settings for validation, press the OK button to return to the UPG work environment. If you want to modify the default validation settings, please review the next section for more details.

Configure Validation Options

UPG allows you to change several settings that modify the behavior of data validation. Most notable among these settings is to provide your users with the ability to override validation. If an entered value is not validated, the user can choose to accept the value anyway. Please complete the following steps to modify your UPG validation settings:

1 **Enable the following checkboxes according to the descriptions provide below:**

□ **Ignore if File Missing**. Enable this checkbox if you want your UPG application to ignore any Lookup Field validation if the specified validation file is missing (this will enable your application to operate as if not validation is defined). If this checkbox is disabled, then your application will require a proper validation file to operate correctly.

- □ **Allow Override**. Enable this checkbox to allow a user to enter a value that is not contained in the validation file. If this checkbox is disabled, the user will only be able to enter values that are contained in the specified validation file.

2 **Press the OK button to return to the UPG work environment when you have completed modifying your settings.**

Completing Validation

Once you have configured your input fields for validation, you will need to make sure that you have a method for placing validation files onto your portable data collection unit.

The easiest way to place validation files onto your portable is to build the necessary Receive Data from Host utilities. In other words, make an individual utility for each required validation file.

For more information on creating utilities, please see Chapter 4, Using UPG. The Add Utilities section will contain all of the necessary steps required to add a utility to your portable data collection application.

Input Source

UPG provides you the ability to control the method of data entry on an individual input field basis. For example, you can require keyboard entry for a user password, a scanner (input devices such as a wand or laser) for an item number, or any source for quantity.

This is a useful feature if you want to “force” your users to enter data from barcodes only; this can easily be achieved by selecting scanner from the Input Source pull-down list.

Tip: Another method of improving the accuracy of your data collection operations is to require that certain pieces of data only be entered via a scanner. The accuracy of a barcode and scanner is far greater than manual data entry.

Validation is defined at the input field level. In the previous chapter, you learned how to define an input field when building a multipage sequence.

1 **Open the multipage sequence that contains the input field that you want to apply the Input Source modification to.**

2 **Make sure that you have the input field definition screen open for the input field you want to modify.**
You can specify an Input Source when you first add an input field to a multipage sequence, or you can modify the Input Source value of an existing input field.

If you want to modify the Input Source value of an existing input field, complete the following to edit an input field:

a **Position your cursor in the Screen window, right-click, and select Edit Control.**

b **Select the input field that you want to edit from the selection box.**

3 **Press the Verify tab to access the Input Source pull-down list.**
All data verification functions, in fact, are contained within the Verify tab of an input field definition screen.

4 **Select the desired input source from the Input Source pull-down list based on the following descriptions:**

❑ **Any**: Will accept data input from either the portable's keyboard or scanning device (such as a wand, laser, etc.).

❑ **Keyboard**: Accepts data from the portable's keyboard only; data entered via the scanner will not be accepted and the user will receive the message, “Keyboard Input Only.”

❑ **Scanner**: Accepts data from the scanning device attached to your portable only; data entered via the keyboard will not be accepted and the user will receive the message, “Scanner Input Only.”

5 **Press the Ok button when you have completed selecting your data input source. You will be returned to the multi-page sequence definition screen.**

Entry Patterns

Entry Patterns are another method available to you for increasing the accuracy of your data collection operations. For example, if you wanted the entered value to match the U.S. Social Security Number format, you would use the entry pattern ###-##-#### to enforce such a requirement.

More advanced entry pattern features (see Advanced Entry Patterns later in this chapter) allow you to parse data, define multiple entry pattern for one input field, and make special patterns depending on the input device (you can define an entry pattern unique to scanned input and another entry pattern specific to a keyboard entry).

Validation is defined at the input field level. In the previous chapter, you learned how to define an input field when building a multipage sequence.

1 **Open the multipage sequence that contains the input field that you want to apply an entry pattern to.**

2 **Make sure that you have the input field definition screen open for the input field you want to modify.**
You can specify an entry pattern when you first add an input field to a multipage sequence, or you can modify the entry pattern of an existing input field.

If you want to modify the entry pattern of an existing input field, complete the following to edit an input field:

a **Position your cursor in the Screen window, right-click, and select Edit Control.**

b **Select the input field that you want to edit from the selection box.**

3 **Press the Verify tab to access the Entry Pattern field.**

Please see the following section to learn how to apply the different capabilities of Entry Patterns to your input field.

Simple Entry Patterns

UPG allows you to implement pattern matching using a series of character “building blocks.” Assemble the building blocks in the order that you want your entry pattern to match. Any entry that does not match the entry pattern will be rejected and the user will receive the following message: “Entry Does Not Match Specified Pattern.”

Basic Entry Pattern Building Blocks

Character	Description	Example
0-9	Entry pattern will accept any of the specified numbers.	146 (accept only the number 146). 5#### (accept any 5 digit number that begins with 5)
a-z, A-Z	Entry pattern will accept any of the specified alpha characters. The entry pattern is case insensitive until a $ (see below) is used.	akdt (accept only akdt as a valid entry, with any combination of character case)
$	Make entry pattern case sensitive. An entry pattern is case insensitive unless a $ is used. If $ is used more than once in an entry pattern, the case status is toggled.	$abcdEFG (accept only abcdEFG as a valid entry).
?	Entry pattern will accept any single character; operates like a single character wildcard.	##?## (accepts 2 numbers, followed by any single character, followed by 2 numbers).
#	Entry pattern will accept only numerical data (0-9).	### (accepts 3 numbers)
@	Entry pattern will accept an alpha character (is not case sensitive).	@@@@ (accepts 4 letters)
:	Entry patterns will accept either alpha or numeric data entry.	:::: (accepts any combination of 4 numbers or letters)
^	Will only accept uppercase alpha data entry.	^^^ (accepts only 3 uppercase letters)
&	Will only accept lowercase alpha data entry.	&&& (accepts only 3 lowercase letters)
*	Entry pattern will accept any number of characters after the wildcard (*). This works just like the wildcard (*) in a DOS command, such as `DIR t*.*`. You should only place a wildcard at the end of a pattern.	###* (accept 3 numbers followed by any number of valid characters, such as 333kd-9 or 333h-9-kk)
’	This is the escape character to be used if you want to include a building block character in your entry pattern.	’###-## (accepts a number sign # followed by 2 numbers, a dash, and 2 more numbers, i.e. #33-44)

Table 5-2: Combine the individual building block characters in this table to form meaningful entry patterns; an entry pattern can increase the quality of collected data by ensuring that only data with the proper format is entered.

Please complete the following steps to add a simple entry pattern to the currently selected input field:

1 **Assemble the character building blocks in the Entry Pattern field until you have completely built your entry pattern:** The best way to illustrate how to define an entry pattern is to provide several popular examples. If you would like, you can insert the following samples into your portable data collection application:

❑ Social Security: (i.e. 554-32-2234) `###-##-####`

❑ Phone Number: (i.e. (203)555-1234) `'(###')###-####`

❑ Customer ID: (i.e. C4432-8B) `@####-#@`

2 **When you have finished, press the OK button to return to the multipage sequence that contains the currently selected input field.**

Advanced Entry Patterns

In addition to simply enforcing simple entry patterns, you have the ability to manipulate the collected data with advanced entry pattern building blocks. Table 5-3 lists all of the advanced entry pattern building blocks. Each is discussed in detail in the following sections.

Advanced Entry Pattern Building Blocks

Character	Description	Example
(*zone number*)	Match Zones: Enclose match zone numbers within parenthesis.	(3)###(1)@@(2)## (will accept 3 numbers, followed by 2 letters, followed by 2 numbers as valid data entry, yet will save data as 2 letters followed by the second group of numbers followed by the first group of numbers)
‘	Requires scanned data input (will accept data entry via a laser, CCD, wand, etc.); add open single quote (‘) to the end of the entry pattern.	###‘ (will accept 3 numbers from scanned input only)
~	Required data input from the portable keypad; add the tilde (~) to the end of the entry pattern.	###~ (will accept 3 numbers from the keyboard only)
/*literal string*/	Allows you to enter a literal string into your collected data; this is helpful when you are required to add a suffix or prefix to the collected data.	/PN/### (accepts 3 numbers as valid data entry, but adds PN as a prefix when data saved to disk)
<*target field*>	Re-directs where collected data is saved based on the entry pattern used; useful only when you have defined multiple entry patterns for a single input field.	<Entry01>###|<Entry02>@@@ (if three number are entered, data saved to input field Entry01 and if 3 letters are entered, data saved to input field Entry02)
=	Allows you to substitute a collected character with a character of your choosing.	A=C### (will accept A followed by 3 numbers as valid data entry, but will save C followed by the 3 entered numbers to disk)
{*saved data*}	Enclose the portions of the entry pattern that you want to save to disk; the portion of the entry pattern outside the braces will be discarded (used for parsing).	##{###}# (accepts a 6 digit number as valid data entry, but only saves the third, fourth, and fifth numbers to disk)
+[*pad length*], +0[*pad length*]	Use these building blocks to specify space and/or zero padding. Use the syntax *+[pad length]* or *+0[pad length]* to specify the length of your pad; should be placed at the end of your entry pattern.	###+6 (accepts 3 numbers as valid data entry and pad s spaces before the 3 entered numbers) ###+06 (same as the above, but pads 3 zeros followed by the 3 entered numbers, i.e. 000456)
[*character set*]	Enclose the valid character set within brackets for a position ([]); use a hyphen (-) to specify a range of characters.	[abc]## (accepts only a, b, or c followed by 2 numbers as valid data entry)
^	When placed at the beginning of a character set, makes the values in the character set invalid.	[^abc]### (accepts all letters except a, b, or c followed by 3 numbers as valid data entry)

Table 5-3: Use the advanced building block characters contained in this table to increase the power of your entry pattern. Many of these building blocks not only ensure quality of the data entered, but also allows you to manipulate the entered data.

Character Sets

Previously, you have learned how to require a letter (case sensitive or insensitive) in an entry pattern. Defining a character set will allow you to define a range of characters that are acceptable data entry for a particular entry pattern position.

For example, you might want to accept a 3 digit number that has a prefix within the range of a-e. You would enter the value [a-e]### as your entry pattern. You can also use a caret (^) to exclude ranges. For example, you might enter [^abcd] to accept data entry that does not include a, b, c, or d in the first character position.

Building Blocks: `[`*character set*`]`, `^`

Examples: `[A-M]###`

Will accept data entry of `A` *through* `M` *followed by three numbers.*

[^abc]##

Will accept data entry of any letter except `a`, `b`, *or* `c` *followed by two numbers.*

[cgkm]@@@

Will accept 4 letters, but the first character must be `c`, `g`, `k`, *or* `m`.

Data Parsing

It may be necessary at times to remove certain characters that are contained in a barcode (or collected data for that matter); for example, if the value of a barcode is ISBN-67543, you may want to write only the “67543” to your data collection file. You can use data parsing to remove the “ISBN-” portion of the barcode. This is a very simple example, but UPG’s data parsing capabilities (sometimes called substring extraction) allow you to remove any part of the collected data, even if it is in the middle of a value.

Braces, sometimes known as *curly brackets*, ({}) are used to surround the portion or portions of the entry pattern that you want to save; all portions of the entry pattern that are not within the braces will not be saved to disk (it will be discarded).

Building Blocks: {*saved data*}*discarded data*

Examples: ##{###}@@

Will accept data entry of 5 numbers, followed by 2 letters. Only the third, fourth, and fifth numbers will be saved, while the rest will be discarded.

##{@@}#{##}

Will accept 2 number, followed by 2 letters, followed by 3 numbers as valid data entry. Only the 2 letters and the last 2 numbers of the entry pattern will be saved; everything else will be discarded.

Padding Data

Padding data allows you to fill in the extra spaces of a field with either zeros (0) or spaces; the spaces or zeros are added to the left of the entered text. For example, if you specify zeros to fill up to 10 characters and only a 4 character value is entered, the value written to the field would be `000000NNNN`. You will most often use data padding in conjunction with fixed length fields.

To specify data padding, you add a building block to the end of your entry pattern. Use a Plus sign and a 0 (+0) for zero padding and just a Plus sign (+) for space padding. Specify the amount of characters you want either the spaces or zeros to pad up to after the +0 or + sign. For example, if you want to pad the input field with zeros up to 10 characters in length, you would add +010 after your entry pattern.

Building Blocks: `+0`*zero pad length*, +*space pad length*

Examples: `###+015`

Accepts 3 numbers as valid data entry and pads the field with zeroes up to 15 characters in length; for example, if `233` *was entered,* `000000000000233` *would be saved to disk.*

@@@+13
Accepts 3 letters as valid entry and space pads the field up to 13 characters in length. For example, if `TRR` *was entered, 10 spaces followed by* `TRR` *would be saved to disk.*

Pattern Literals

In special circumstances, you may want to add a literal string to your collected data. For example, you might be required to add a prefix or suffix to your collected data.

Also, you might want to use pattern literals to identify different types of data. Specifically, since you have the ability to define multiple entry patterns for a single input field, you might want to use a literal string to identify which entry pattern was used. You may also want to use a pattern literal to identify the input source of the collected data.

Building Blocks: `/`*literal string*`/`

Examples: `/ABC/###`
Accept 3 numbers only; will add `ABC` *as a prefix to the 3 collected numbers.*

`###/-R/`
Accept 3 numbers only; will append `-R` *as a suffix to the 3 collected numbers.*

Character Substitution

UPG allows you to make single character substitutions within your entry patterns. For example, if you had the entry pattern T###, you could substitute the prefix T with a character of your choosing, such as R.

The building block used for character substitution is the equals sign (=). If you wanted to make the substitution described above, you would enter T=R### in the Entry Pattern field.

Building Blocks: *entered character=substitute character*

Examples: A=C###

Will accept data entry of A *followed by three numbers, but will save the collected data as* C *followed by the same three numbers.*

###T=Q

Will accept 3 numbers followed by a T as valid data entry, but will instead save to disk the same three numbers followed by a Q.

Source Processing

Source Processing allows you to differentiate between data entered with a scanner and a keyboard. This ability allows you to apply a different entry pattern upon data entered with either a keyboard or scanner.

Use an open single quote (‘) to identify scanner input and a tilde (~) to identify keyboard input. The building block is placed at the end of the entry pattern; for example, to accept 3 numbers followed by 2 letters from the keyboard only, you would put ###@@~ as your entry pattern.

Building Blocks: ‘ *and* ~

Examples: #####‘

Accept 5 numbers as scanned input only.

#####~

Accept 5 numbers as keyboard input only.

Multiple Entry Patterns

If you want to define multiple entry patterns for your input field, simply separate the entry patterns with a pipe (|). For example, if you wanted to define #### and ##@## as the multiple entry patterns, you would specify ####|##@## in the Entry Pattern field.

Building Blocks: **|** (pipe)

Examples: `@@@@~|####'`
Accept 4 letters from keyboard and 4 numbers from scanned input.

`/K/###~|/S/###'`
Accept 3 numbers only; will append `K` *as a prefix if data entered with the keyboard. Will append* `S` *as a prefix if data is scanned input (remember to use a pipe (|) to separate multiple entry patterns).*

Field Selection

At times you may want to place collected data in different input fields based on the entry pattern of the data collected. For example, if you had `a###|b###` as your defined entry patterns, this input field will accept data entry of 3 numbers preceded by either an `a` or a `b`. Using the Field Selection building blocks, you can modify this entry pattern so that data entry with the data identifier `a` is saved in the current input field, while data entry with the data identifier `b` is saved in another input field.

How could you use this advanced feature in your portable application? Consider the following scenario: Part Number barcodes have the data identifier P while your Serial Number barcodes have the data identifier S. Periodically, the wrong barcode is scanned for the input field (Serial Numbers, for instance, are often scanned as Part Numbers); this situation can be a major cause of inaccurate data collection. Using the Field Selection building blocks, you can ensure that the proper barcode data is entered into the correct input field.

Specifically, let's say the part number input field name is `entryPartNumber` and that the serial number input field name is `entrySerialNumber`. You would enter `<entryPartNumber>P#####|<entrySerialNumber>S#####` in the **ENTRY PATTERN** field for both input fields. When either the P##### or S##### barcode is scanned, this entry pattern will ensure that the entered data is saved to the respective field.

Collected data can be saved to different input fields contained within the same multipage sequence. It is not possible to collect data into one multipage sequence and then re-direct the collected data to an input field contained in another multipage sequence. Also, using field selection building blocks only makes sense if you have allowed for multiple entry patterns. If you have only identified one entry pattern, the collected data should then be saved to the current input field.

Building Blocks: <*target input field name*>

Examples: `<PartNumber>P#####|<SerialNumber>S#####`

Accept 2 different patterns: 5 numbers with prefix `p` *or* `s`*. If the collected data has the* `P` *prefix, the data will be saved to the* `PartNumber` *input field, while collected data with the* `S` *prefix will be saved to the* `SerialNumber` *input field.*

`<Entry01>a###|<Entry02>b###|*`

Accept 3 different patterns: 3 numbers with prefix `a`*, 3 numbers with prefix* `b`*, and anything. Collected data that is 3 numbers in length with prefix* `a` *will be saved to the input field* `Entry01`*, collected data that is 3 numbers in length with prefix* `b` *will be saved to input field* `Entry02`*, while everything else will be saved to the current input field.*

Match Zones

Specifying match zones within your entry pattern will allow you to rearrange collected data before it is saved to disk. This might be helpful in instances where you want to convert a suffix of a barcode to a prefix before saving the collected data to disk.

The building blocks for match zones are the open and close parenthesis; you identify the zone number within. For example, an entry pattern with three match zones might look like `(3)###(2)##(1)@@`. The input field will only accept 5 numbers followed by 2 letters as valid data entry; however, the data will be saved as (1)@@(2)##(3)###. Specifically, if 22255bb was entered into this field, bb55222 would be saved to disk.

Finally, any portion of your entry pattern that is assigned to a match zone zero (0) will not be saved to disk.

Building Blocks: (*zone number*)

Examples: (3)###(2)&&(1)@@@
Collected data will be saved to disk as @@@&&###.

(1)@##(3)&#@(2)::::
Collected data will be saved to disk as @##::::&#
(1)###(0)@@(2)###.
Collected data will be *saved to disk as ######.*

Match Flags

Match flags allow you to dynamically set the entry pattern for a specified input field. Unlike other entry patterns, match flags require the use of UPG Action components. Essentially, you build multiple entry patterns and assign each entry pattern a unique match flag.

Once you have established the multiple entry patterns and assigned the match flags, you will then use UPG Action components to set the appropriate match flag. Most likely, you will use the IF Action component to set your match flag (the flag is set using an expression available in the UPG Expression Builder - see Chapter 8, UPG Actions for more details). For example, you can set the entry pattern based on data entry or user settings.

Up to ten (10) match flags can be assigned for all input fields contained in a given form. For example, if the `Site` and `Location` input fields were contained in the form `CollectData`, you can assign up to 10 match flags for both input fields; you might use 3 match flags for Site and the remaining 7 for Location. You can assign more match flags for input fields in other forms, but you would not be able to define more match flags for `CollectData`.

Building Blocks: %*flag number*

Examples: `####%1|@@@@%2`

If the match flag is set to 1, then the input field will accept 4 numbers as valid data entry. If the match flag is set to 2, the input field will accept 4 alpha characters as valid data entry.

`@@###%1~|###%2'`

If the match flag is set to 1, then the input field will accept 2 alpha characters followed by three numbers from the scanner as valid data entry. If the match flag is set to 2, the input field will accept 3 numbers from the keyboard as valid data entry.

Valid Characters

List the characters in this field that the currently selected input field will accept as valid characters. Any character entered that is not in this list will immediately be rejected. If you want your alpha characters to be case sensitive, place a $ at the beginning of the list. If there are no values in this field, than all characters will be accepted. (If an entry pattern is defined, entered values will still be subject to pattern requirements).

For example, if you wanted to accept only the first five letters of the alphabet and the numbers 1-5, you would enter `abcde12345` in the Valid Characters field.

1. **Enter the list of valid characters in the Valid Characters field.**
   If you want your list of valid characters to be case sensitive, remember to place a dollar sign ($) at the beginning of your list. If you want to then specify a and A to be valid characters, you will need to place both cases in the valid character list.
2. **When you have finished, press the OK button to return to the multipage sequence that contains the currently selected input field.**

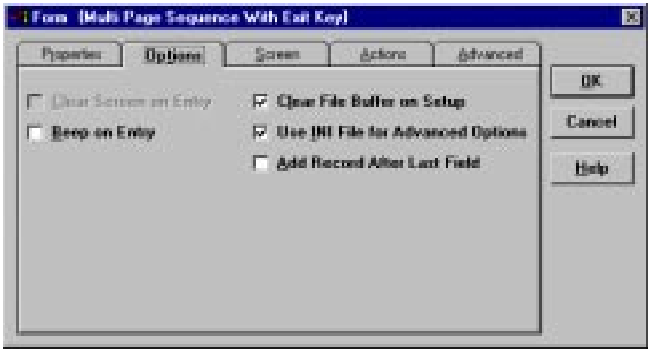
6

Beyond the Basics

Chapter 4, Using UPG, introduced you to the major components contained within the Universal Program Generator. Beyond the Basics will teach you how to harness the flexibility of UPG not covered in Chapter 4. This chapter assumes that you have already created a basic application, but you can also apply the information and steps contained within this chapter as you are building a new application from scratch.

Configure Form Options

Tools created with the Form tooltype have a number of options contained in the OPTIONS tab; these options will enable you to modify the behavior of the Form to suit your individual requirements. The options range from the very simple to very complex. The following section will provide the information necessary for you to implement these options successfully.



Form (Multi Page Sequence With Exit Key)

□ Clear Screen on Entry
□ Beep on Entry
☑ Clear File Buffer on Setup
☑ Use INI File for Advanced Options
□ Add Record After Last Field

Figure 6-1: The Options tab of a Form definition screen allows you to modify the behavior of your Forms, which includes menus, collection sequences, and message boxes.

1 **Double-click on the Form tool that you want to modify or create a new Form tool.**
Please see Chapter 4, Using UPG, for more information on creating new Form tools.

2 **Press the OPTIONS tab.**
Enable the checkboxes that correspond with the options that you want to activate for the currently selected form (see Table 6-1 for details on the individual options).

Options	Descriptions
Clear Screen on Entry[1]	When available, this feature will blank out the screen when a page is reached (works just like the CLS DOS command).
Beep on Entry[2]	If activated, this option will cause your portable unit to produce an audible tone when the form is first opened.
Clear File Buffer on Setup[3]	Clears out all data in the file buffers and resets all values to null.
Use INI File for Advanced Options[3]	If this feature is enabled, an application.ini file will be created for your portable application. For example, if you have a program that is named Collect.exe and this feature is enabled, you will also generate a Collect.ini file the first time you activate your multipage sequence. The user can edit this file to modify sequence order and maximum field lengths.
Add Record After Last Field[3]	After the last field in a sequence has accepted a value, the record will then be added to the target file.

[1] Applies to Message Box, View File, and Multipage Sequence templates.

[2] Applies to Menu, Message Box, View File, and Multipage Sequence templates.

[3] Applies only to the Multipage Sequence template.

Table 6-1: Options available in the OPTIONS tab when defining a *form* tool. Please note that not all options are available on all types of forms.

3 **Press the OK button when you have completed modifying your Form tool.**
You will be returned to the UPG work environment. Build and run your application to view the results of your modifications.

Use INI File

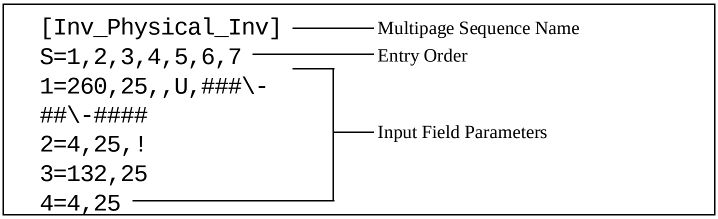
If you enable the Use INI File for Advanced Options checkbox, you and/or your users will be able to modify multipage sequences from the `application.ini` file (for example, if the name of your application is `inv.exe`, than you will be able to use `inv.ini` to modify the behavior of your multipage collection sequence).

Specifically, you are able to configure the following characteristics of your multipage sequence from the `application.ini` file:

- ❑ Entry Order
- ❑ Parameter Value

- □ Maximum Input Length (can be double max length specified in the File/Variable tool)
- □ Default Value
- □ Data Identifier
- □ Entry Pattern
- □ Valid Characters

Entries will be made in the `application.ini` in the following manner: (1) The multipage sequence name will be enclosed in brackets ([]) identifying the individual multipage sequence in the INI file, and (2) the multipage sequence name will be the entry order, followed by the individual settings for each input field (see Figure 6-2 on page 6-4 to see an example).



```
[Inv_Physical_Inv]
S=1,2,3,4,5,6,7
1=260,25,,U,###\-
##\-####
2=4,25,!
3=132,25
4=4,25
```

Figure 6-2: Application.ini file contents. Modify the INI file settings of the desired multipage sequence to modify the behavior of your data collections sequence.

Using the application.ini file to modify your UPG generated application is a great way for you or your user to customize the data collection application without having to modify an application's code and recompiling the application. Simply use any text editor to modify the application.ini file, either on the host PC or on the portable data collection unit. It might be easier to copy the application.ini file from your portable to the host PC, modify the file, and then copy it back to your portable unit; it may be quite tedious to modify the INI file with your portable's screen limitations.

For more detailed information, please see Appendix D, UPG INI Files.

Prompt Substitution

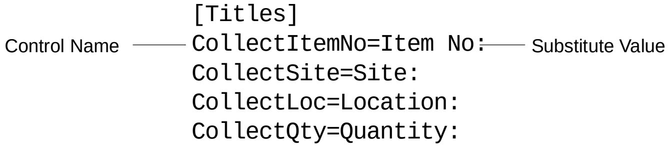
Many users of portable data collection applications like to modify the prompts (or display text) of user interface controls (i.e. input fields, hotkeys, etc.) to values they understand or are appropriate to their individual situation.

For example, you might build a portable application that collects Site, Location, and Item Number. One of your users, however, may want to collect data for Building, Room, and Part. Prompt substitution will allow your users to customize your portable application to fit their needs without having to write a new or modify an existing application.

With this in mind, UPG was developed to allow users to easily substitute display text at the INI file level. As a UPG developer, however, you can provide a mechanism within your portable application to automate the prompt substitution process.

INI Structure

Before you begin building the INI structure to allow your users to change prompts from within the UPG generated application, please review the APPLICATIONP.INI File section on page D-11 in Appendix D.



```
[Titles]
CollectItemNo=Item No:
CollectSite=Site:
CollectLoc=Location:
CollectQty=Quantity:
```

Figure 6-3: Sample APPLICATIONP.INI file; the value on the left is the name of the user interface control and the value on the right is the value that will be substituted for the default value.

To allow your users to change prompts from within the portable application, you will need to build a multipage sequence that will change the prompt or display text for each of the prompts. Logically, if you need to build a multipage sequence, you need a file to store the collected data to.

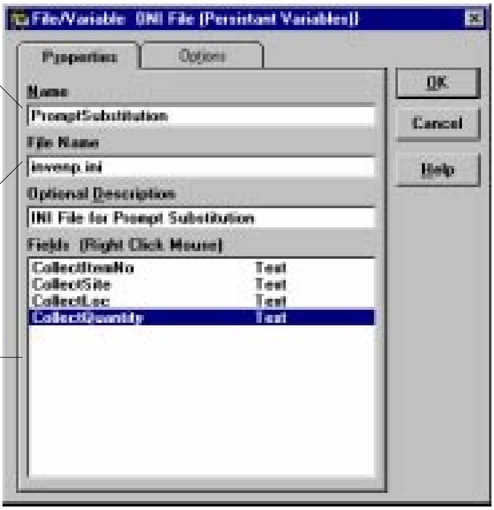
In this scenario, you need to build a persistent INI file that contains fields that are identically named to the name of the controls you are substituting values for.

1 **Before you begin building the APPLICATIONP.INI file, you need to make a list of all of the names of the user interface controls that you want to substitute prompts for.**
As illustrated in Table 6-3, prompt substitution is based on the name defined for each of the user interface controls in your portable application. Before you begin building the APPLICATIONP.INI file, it is important that you make an accurate list of all of the control names that will support prompt substitution. The list that you put together will determine the name of the fields in your APPLICATIONP.INI file.

You may also need to write down what the default prompt (display text) should be for each control. This will usually be what the display text was entered as when the control was created.

2 **Double-click on the FILE/VARIABLE tooltype in the ToolBox.**
You can also right click on the FILE/VARIABLE tooltype in the TOOLBOX and select ADD.

3 **Highlight the INI FILE template and press the OK button.**
Creates a file structure similar to a memory variable file, but stores the information between application sessions (data is not lost when the application is terminated). Using an INI File will ensure that your substituted values remain intact between application sessions.



Name
PromptSubstitution

File Name
invenp.ini

Optional Description
INI File for Prompt Substitution

Fields (Right Click Mouse)

```
CollectItemNo          Text
CollectSite            Text
CollectLoc             Text
CollectQuantity        Text
```

The value in the **Name** field is how you will refer to this file within the UPG development environment.

The value in the **File Name** field is the DOS name of your file.

Your field names should be listed exactly as your controls are named; this will ensure that your prompts substitute correctly.

Figure 6-4: Sample APPLICATIONP.INI file.

4 **Enter `PromptSubstitution` in the NAME field and press <TAB>.**

This is the name that you will refer to while programming within the UPG environment (this is not the name of the DOS file that will be placed on your portable; see step 5).

Note: Whenever you name a tool (using the NAME field), your entered value must comply with C programming naming conventions. Any invalid character will be converted to an underscore.

5 **Enter the DOS file name of your APPLICATIONP.INI file in the File Name field. Press <TAB>.**

You will notice that a DOS file name will automatically be created from the tool name entered in step 3. In order for prompt substitution to work correctly, you must name your APPLICATIONP.INI file in the following manner:

[application name]p.ini

For instance, if the name of your application was inven.exe, you would name your APPLICATIONP.INI file `invenp.ini`. If your APPLICATIONP.INI file is not named in this manner, prompt substitution will not work correctly.

6 **Place a description for your file in the OPTIONAL DESCRIPTION field.**

This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

7 **Press <CTRL><B> to Batch Add fields to your data file.**

You will add two fields to the UserVerification file. Using Batch Add will let you quickly add to fields to your file.

8 **Enter the control name from your list in the NAME field.**

The field name MUST be entered exactly as the control is named; if there is any variation, your prompt substitution may not work correctly.

9 **Enter `Titles` in the Section field.**

You prompt substitution fields will be listed under the heading `[Titles]` in your APPLICATIONP.INI file.

10 **Leave Text selected as the Field Type.**

11 **Enter a maximum entry length suitable for the target portable in the Max Length field.**

While you can specify an supported length that you would like, remember that your portable display is quite small. You should make the maximum length suitable for your portable.

If you are building applications for multiple portables, specify the smallest width as your max length. This will ensure compatibility between the different supported portables.

12 **If desired, enter a description in the OPTIONAL DESCRIPTION field.**

This field is for documentation purposes only and exists so that you may include descriptions for yourself and other developers. Any description placed in this field will not be included in the UPG generated source code; your descriptions will only be saved in the current toolbox file.

13 **Enter the default prompt (display text) in the DEFAULT field.**
If the APPLICATIONP.INI file is deleted from the portable, your portable application will automatically create a new APPLICATIONP.INI file with the defaults specified in this field.

14 **Press the OK button.**
After you press OK, a second field definition screen will appear.

15 **Another Input Field definition screen will appear. Repeat steps 8-15 until you have completed adding all of the controls on the list that you created in step 1.**
When you have completed defining all of the necessary input fields, press Cancel on the Input Field definition screen that appears after you have defined the last necessary input field; this will stop the Batch Add Input Fields process.

16 **Press the OK button on the File/Variable definition screen to return to the UPG work environment.**

Change Prompt Utility

While it is possible for you or your users to change the prompts by editing the APPLICATIONP.INI file using a text editor, as a developer you may want to provide a utility within the application to change the prompts. The process is simple: you will create a multipage sequence that changes the values contained in the APPLICATIONP.INI file. When you have finished creating the utility, be sure to link it to a menu using a hotkey.

Please complete the following steps to create a multipage sequence to allow your users to modify the prompts from within a UPG application.

1 **Double-click on the Form tooltype in your ToolBox.**

2 **Highlight Multipage Sequence and press the OK button.**
Since changing user prompts requires data entry, you will need to use the Multipage Sequence Form template to build the tool that will change your user prompts.

3 **Enter `ChangePrompts` in the Name field.**

This is the name that you will refer to while programming within the UPG environment.

4 **Enter `Change Prompt Screen` in the OPTIONAL DESCRIPTION field.**

Recall that this field is for documentation purposes only.

5 **Select `PromptSubstitution` from the FILE pull-down list.**

6 **Press <CTRL><B> to add input fields to your `ChangePrompts` multipage sequence; select Input Field.**

7 **Enter the same control name in the Name field as the name of the user interface control you are substituting prompts for.**

If you name this input field the same as the control you are substituting prompts for, then this input field will also have the prompt (display text) substituted. Why is this helpful? When your user changes prompts, they will always be able to see what the current prompt is.

8 **Select the corresponding field from the INPUT FIELD pull-down list. You can easily identify the set of corresponding fields because they will be preceded by `PromptSubstitution`.**

When your users enter a value into this input field, the value will be saved to the field you select from the INPUT FIELD pull-down list. This saved value will be the substituted prompt.

9 **Enter a value in the DISPLAY TEXT field that will allow your users to identify which field they are changing.**

In most cases, it would be acceptable to enter the default display text for the target input field in the DISPLAY TEXT field.

10 **Repeat steps 7-10 until you have added all of the necessary input fields.**

If you adopted the naming syntax described in step 7, you should easily be able to correlate the input fields in the ChangePrompts multipage sequence to the list of controls you created in the previous section.

11 **Press the Cancel button when you have completed defining all necessary input fields.**
This will stop the Batch Add Input Fields process. You will be returned to the multipage sequence definition form.

12 **Press the OK button to return to the UPG work environment.**

Note: Remember to add a hotkey to call the **`ChangePrompts`** tool somewhere within your portable application. Please see Chapter 4, Using UPG for more information on adding/using a hotkey.

Customize Data Entry

Using several optional input field settings, you can easily modify the behavior of your data collection sequences. Such settings can modify the way data identifiers are used by defining default values (which can save your users time), changing the location of the input field, and modifying the enabled status.

To modify the optional settings for your input fields, please complete the following:

1 **Double-click on the multipage sequence that contains the input field that you want to modify.**
If you are creating a new multipage sequence, please see Chapter 4, Using UPG for instructions on creating a new multipage sequence.

2 **Double-click on the input field that you want to modify.**
You can also modify the optional settings as you are adding an input field to a multipage sequence.

3 **Press the Options tab of the input field definition screen.**
Modify the default settings according to the descriptions contained in the following three sections: Using Default Values, Data Identifiers, and More Options.

Using Default Values

Default values can increase the efficiency and convenience of a portable data collection application. Using a default value means one less piece of data to collect, and default values are often defined for pieces of data that are repeatedly collected between data collection sequences. Please complete the following steps to define a default value for the currently selected input field.

1 **Enter the default value into the Default Value field.**
The value that is entered into this field will automatically be entered in the input field when the user enters this data collection sequence.

Tip: If you want your default value to be dynamically determined (i.e. to have the default value be the value of another field), use a "bang" (`!`) along with the `file.field` to determine the source of the default value. For example, enter `!InvSettings.Site` to have the value of the Site field of the InvSettings file (usually an INI file) to be used as the default value.

2 **Enable (check) the REUSE DEFAULT EACH TIME checkbox to use the specified default value each time this input field is reached.**
As the user loops through your data collection sequence, each time the currently selected input field is reached, the default value specified in step 1 above will be automatically entered.

If this checkbox is disabled, the default value will be automatically entered the first time the currently selected input field is reached, but not during subsequent passes through the multipage sequence.

Note: As mentioned in the Use INI File section earlier in this chapter, the default value is contained in the application.ini file. The value that is defined in the Options tab of the input field definition form can be modified from the application.ini file. Please review the Use INI File section earlier in this chapter and Appendix D for more information.

Data Identifiers

A data identifier is usually a string of characters that precede a bar code that identifies the type of data; for example, a bar code may have the value "`PN123-76`." The data identifier or prefix "`PN`" identifies that this bar code is a part number.

1 **Type the desired prefix in the DATA IDENTIFIER field.**
Using the example above, enter “`PN`” in this field. If you want to dynamically determine the data identifier, you can specify the field to get the identifier from by using the “bang” along with the `file.field` name; for example, you might enter `!InvSettings.DataIdentifier` to retrieve the data identifier from an INI file.

2 **Enable the DATA ID REQUIRED checkbox to require the specified data identifier be included in the entered data.**
If this checkbox is enabled and the user enters a value that does not contain the specified data identifier, the user will receive the following error message; “`Entry has incorrect data identifier`.” The user will be required to enter a value that contains the data identifier and will continue to receive the previous error message until an entry with the specified data identifier is entered.

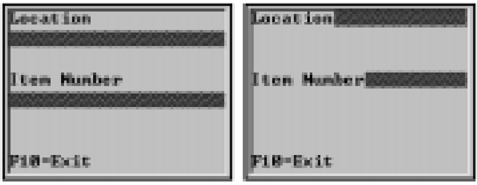
3 **If required, enable (check) the STRIP DATA ID FROM INPUT checkbox.**
In some instances, you may want to remove the data identifier from the entered data before it is written to the target file. For example, if the data identifier is “`PN`” and the user enters “`PN123-45`” and this checkbox is enabled, only “`123-45`” will be written to the target file.

More Options

There are several more options available to customize your data entry. Please configure these settings according to the following steps/descriptions.

1 **Select the input field position from the FIELD POSITION pull-down list.**
The field position is determined in relationship to the display text (review Table 6-5 below for more information).

Next Line: Input Field is placed below the entered display text.



Same Line: Input Field is placed to the right of the entered display text.

Figure 6-5: Specifying Same Line will allow you to place more input fields on one page. If you specify Same Line, however, consider the length of data entry; the visible data entry will be less than if you specify Next Line. Compare the field lengths between the two screens above: Next Line on the left has a much larger input field than the Same Line on the right. If the field maximum size is greater than the input field size, you will still be able to enter data up to the maximum size, though your user will not be able to see the entire entered value.

2 **Select the desired character case from the CASE pull-down list.**

The value entered in the currently selected field will be entered as the case that you select in this pull-down list. For example, if UPPER CASE is selected, any entered value will be entered in upper case letters.

3 **Enable the DISABLED BY DEFAULT checkbox if you want the currently selected input field to default to the disabled state.**

If enabled, the currently selected input field will not be able to receive data entry until this input field is enabled. To enable this input field, use the Enable/Disable UPG Action component to enable this field.

You might ask yourself, “Why would I add an input field to my multipage sequence and then disable it by default?” You may not want to have data entered into this input field, except for certain circumstances. For example, you may not want to collect Site data by default (the default value will be automatically entered). But if a user presses a hotkey to enter Site data, you can activate the Site input field by using a hotkey in conjunction with an ENABLE/DISABLE UPG Action component.

4 **Enable the DISABLE AFTER ENTRY checkbox to disable the currently selected input field after a value is entered.**

This feature works very similar to the DISABLE BY DEFAULT feature, but the input field is disabled after a value is entered into the current input field during the first pass through the multipage sequence.

Use an ENABLE/DISABLE UPG Action component to re-enable this input field.

Implement Security

With UPG, you can implement a basic user security system on your portable data collection application. This system will consist of a User ID/Password validation file, User ID/Password variables, turning on the password option, and restricting users from entering DOS (except for those with valid Admin access).

Please complete the following to implement basic system security.

Build User Validation File

1 **Double-click on the File/Variable tooltype.**
Before you can build the password/login validation screen, you need to build the data structure to store your User ID and password information.

2 **Highlight ASCII File and press the OK button.**

3 **Enter `UserVerification` in the Name field.**
This will be the tool name of your User ID/Password validation file.

4 **Enter `UserPswd` in the File Name field.**
A File Name will automatically be generated from the value entered in the Name field; overwrite it with `UserPswd`. You do not need to enter the .txt extension to your file; it will automatically be added for you.

5 **Enter `User ID/Password Validation File` in the Optional Description field.**
This description is for documentation purposes only and exists only to help you and other developers identify the purpose of this file.

6 **Press <CTRL><B> to Batch Add fields to your data file.**
You will add two fields to the UserVerification file. Using Batch Add will let you quickly add to fields to your file.

7 **Enter `UserID` in the Name field for the first field.**

8 **Leave Text selected as the Field Type.**

9 **Enter `25` in the Max Length field.**
This will limit the User ID entry to 25 characters or less in length.

10 **Enter `User ID Validation Field` in the Optional Description field.**

11 **Press the OK button.**
After you press OK, a second field definition screen will appear.

12 **Enter `Password` in the NAME field.**

13 **Leave Text selected as the Field Type.**

14 **Enter 8 in the Max Length field.**
This will limit your passwords to 8 characters or less. You will be able to define a minimum password length when you define the password input field in the login multipage sequence.

15 **Enter `Password Validation Field` in the Optional Description field.**

16 **Press the OK button.**
Another blank field definition screen will appear.

17 **Press the Cancel button to return to the File/Variable definition screen.**
Pressing Cancel will stop the Batch Add process.

18 **Press the OK button on the File/Variable definition screen to return to the UPG work environment.**

Once you have completed building your User ID/Password validation file, you are now ready to begin building the User Login screen.

Build Work Variables

1 **Double-click on the File/Variable tooltype.**
Before you can build the password/login validation screen, you need to build the data structure to store your User ID and password information.

2 **Highlight Memory Variables Group and press the OK button.**

3 **Enter `UserLoginWorkVariables` in the Name field.**
This will be the tool name of your User ID and password work variables file.

4 **Enter `User ID/Password Work Variables` in the Optional Description field.**
This description is for documentation purposes only and exists only to help you and other developers identify the purpose of this file.

5 **Press <CTRL><B> to Batch Add fields to your data file.**
You will add two fields to the UserLoginWorkVariables file; using Batch Add will let you quickly add to fields to your file.

6 **Enter `UserID` in the Name field for the first field.**

7 **Leave Text selected as the Field Type.**

8 **Enter `25` in the Max Length field.**
This will limit the User ID entry to 25 characters or less in length.

9 **Enter `User ID Collection Variable` in the Optional Description field.**

10 **Press the OK button.**
After you press OK, a second field definition screen will appear.

11 **Enter `Password` in the NAME field.**

12 **Leave Text selected as the Field Type.**

13 **Enter `8` in the Max Length field.**
This will limit your passwords to 8 characters or less; you will be able to define a minimum password length when you define the password input field in the login multipage sequence.

14 **Enter `Password Collection Variable` in the OPTIONAL DESCRIPTION field.**

15 **Press the OK button.**
Another blank field definition screen will appear.

16 **Press the CANCEL button to return to the File/Variable definition screen.**
Pressing Cancel will stop the Batch Add process.

17 **Press the OK button on the File/Variable definition screen to return to the UPG work environment.**

Once you have completed building your User ID/Password work variables, you are now ready to begin building the User Login screen.

Build User Login

The User Login screen will consist of two input fields on a one page multipage sequence. The entered data will be validated against the User ID/Password validation file and asterisks (*) will be echoed to the screen as the user enters the password. Once the User Login screen has been built, you will attach it to your startup tool. When your portable data collection application is launched, your users will be required to enter a valid user ID and password.

1 **Double-click on the Form tooltype in your ToolBox.**
Your login screen will be built using the Form tooltype.

2 **Highlight Multipage Sequence and press the OK button.**
Since the login screen will be accepting data entry, you must use the Multipage Sequence template.

3 **Enter `UserLogin` in the Name field.**
This is the name that you will refer to while programming within the UPG environment. It is especially important that you specify a descriptive name, such as `UserLogin`, for the login screen since you will be calling this form from a UPG Action component.

4 **Enter `User Login Screen` in the OPTIONAL DESCRIPTION field.**
Recall that this field is for documentation purposes only.

5 **Select `UserLoginWorkVariables` from the FILE pull-down list.**

6 **Remove the `F10=Exit` from the login screen.**
You do not want your user to exit from this screen without entering a valid user ID and password. Removing this hotkey

will ensure that your users will be required to enter such data. Please perform the following to remove this hotkey:

a **Position your cursor in the Screens window.**

b **Right-click and select SELECT CONTROL.**

A list of all of the controls on the current form will be listed.

c **Select Std_Hotkey_Exit from the pop-up menu.**

You will notice that the F10=Exit control on your screen now appears in red. This means that the control is selected.

d **Press <CTRL><D> to delete the control from the form.**

Press the YES button to confirm that you want to delete the control from the current form.

5 **Press <CTRL><B> to add input fields to your UserLogin multipage sequence; select Input Field.**

6 **Enter `inputUserID` in the NAME field.**
This field will accept data entry for the User ID

7 **Select `UserLoginWorkVariables.UserID` from the INPUT FIELD pull-down list.**
The entered User ID will be stored in the field selected from the pull-down list.

8 **Press the VERIFY tab.**
The Verify tab contains all of the tools necessary for you to perform validation and other verification functions within your UPG generated application.

9 **Select `UserVerification.UserID` from the LOOKUP FIELD 1 pull-down list.**
This will compare the entered value against the values contained in the field specified in this field. If a match exists, the entered value will be valid, otherwise, it will be invalid.

10 **Verify that the IGNORE IF FILE MISSING and ALLOW OVERRIDE checkboxes are disabled.**
Disabling both of these checkboxes will require that the user of your portable application enter valid data.

11 **Press the OK button.**
Another blank input field definition screen will appear.

12 **Enter `InputPassword` in the NAME field.**
This will be the input field that will accept the user password.

13 **Select `UserLoginWorkVariables.Password` from the Input Field pull-down list.**
The entered user password will be stored in the field selected from the pull-down list.

14 **Press the Options tab.**
Many optional settings are available in the Options tab; please see Customize Data Entry earlier in this chapter for more information on the available settings.

15 **Enable (check) the Password checkbox.**
This will echo asterisks (*) to the screen while a password is being entered.

16 **Press the VERIFY tab.**

17 **Select `UserVerification.Password` from the LOOKUP FIELD 1 pull-down list.**
This will validate the entered password against the values contained in the UserVerification.Password field.

18 **Select `UserVerification.UserID` from the LOOKUP FIELD 2 pull-down list.**

19 **Select `UserLoginWorkVariables.UserID` from the LOOKUP SOURCE 2 pull-down list.**
Specifying this secondary lookup will verify that the password entered is the correct password for the entered user ID; if the password does not match the User ID, it will not be a valid entry. Both the password and user ID must match for the password to be correct. If any of the two are incorrect, the password will fail.

20 **Verify that the IGNORE IF FILE MISSING and ALLOW OVERRIDE checkboxes are disabled.**
Disabling both of these checkboxes will require that the user of your portable application enters valid data.

21 **Press the OK button.**
Again, another blank input field definition form will appear.

22 **Press the CANCEL button.**
This will stop the Batch Add process and return you to the `UserLogin` form.

23 **Press the OK button to return to the UPG work environment.**

Attach to Startup Tool

There are several advantages to attaching your login screen to the startup tool. You can force the user to re-login after a predetermined amount of inactivity and you will enforce security from one central point (as opposed to implementing it on many individual tools).

1 **Double-click on your existing startup tool.**
In order for this type of security to work, your startup tool must have been made with the Form tooltype. This will most often be a main menu.

2 **Press the Actions tab.**
This will allow you to add UPG Action components to your UPG generated application. For more information on UPG Action components, please see Chapter 7, UPG Action Components.

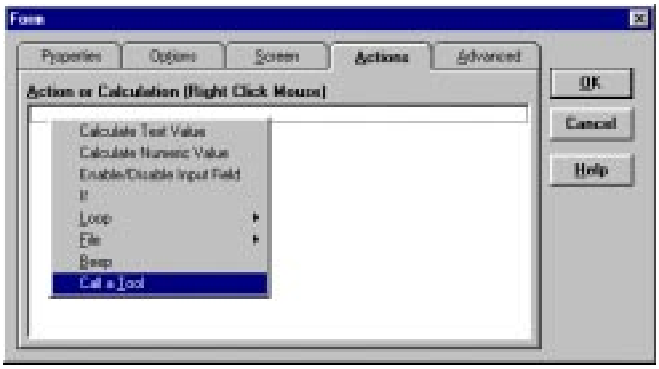


Figure 6-6: When activating a simple security system, a UPG Action component is used to execute the login procedure when your application is started.

3 **Press <CTRL><A>; select the CALL FUNCTION menu item from the pop-up menu (see Table 6-6).**
You may also position your mouse in the ACTIONS OR CALCULATIONS window, right-click, select ADD ACTION, and then select CALL FUNCTION.

4 **Enter `CallUserLogin` in the Name field.**
As always, you should enter a descriptive name for your UPG Action component. In this case, **`CallUserLogin`** identifies that this UPG Action components calls (or executes) the user login screen.

5 **Select SETUP from the WHEN TO EXECUTE pull-down list.**
The Setup selection will execute the “called” function before the currently selected form appears. Specifically, the UserLogin form will execute just before your startup tool (most likely your main menu) appears.

6 **Select UserLogin from the Function to Call pull-down list.**
The Function to Call pull-down list determines which tool is executed; because we want the login screen to appear before the specified startup tool, select UserLogin from this pull-down list.

7 **Press the OK button.**
Notice that your UPG Action component is listed in the ACTIONS OR CALCULATIONS window as `CallUserLogin`; to the left of your UPG Action name, you will also notice the execution time (`Setup`).

8 **Press the OK button.**
You will be returned to the UPG work environment. If you look closely at your program structure in the Program pane, you will notice that the UserLogin tool is attached to your startup tool.

Define Application Timeout

After a certain length of application inactivity, you can automatically set your application to restart. This will require the user of your portable data collection application to re-login to continue using your UPG generated application.

1 **Double click on your Program tool.**
This is easily defined as the tool contained at the top of your program structure and has the `.exe` extension).

2 **Press the Options tab.**

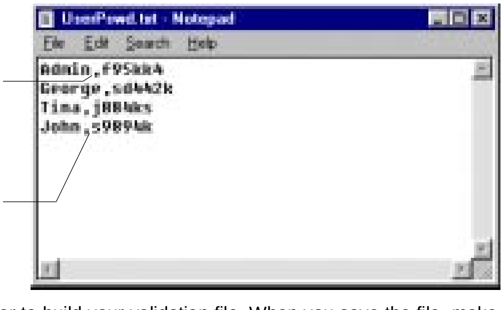
3 **Enter the timeout length in the Timeout field.**
The value entered must be in seconds.

If a key is not pressed within the time limit specified in the Timeout field, your portable application will automatically restart itself. If you have implemented this basic security, a timeout will force your users to re-login after the specified period of inactivity.

The timeout is an essential portion of security, especially if your portable application is installed on a portable that is used by more than one person. Enforcing a timeout will help reduce the amount of unauthorized use. Specifically, somebody else will not be able to simply pick-up a portable that has been sitting around and begin to use the application.

Complete Security

After you recompile, basic security will now be included in your portable application. Since you have added security to your application, however, your portable application will required the proper validation file to work properly. If the UserVerification file does not exist or does not contain valid data, the users of your portable application will not be able enter a valid user ID or password, and will therefore not be able to use your application.



```
Admin,f95kk4
George,sd4k2k
Tina,j88kks
John,s989kk
```

Separate the user ID and the password with a comma (,).

Make sure that you do not place a space ( ) between the comma and the data; if you do, the space will be considered as part of the data.

Figure 6-7: Use a text editor to build your validation file. When you save the file, make sure that it is saved as an ASCII text file and that the file name matches the name specified when you built your validation file (see Build User Validation File).

Make sure that your validation file is a comma delimited ASCII file; this file can be built with a simple text editor, even Windows Notepad.

7 MULTIPLE LANGUAGES & PORTABLES

Percon's Universal Program Generator provides you with the capability to develop a single application that supports multiple hardware platforms and languages. This flexibility will mean less time spent developing and maintaining separate applications.

Developing applications for multiple platforms/languages is quite simple. The first step is to build your "base" application. When your base application is done (consider it done when it contains all of the functionality and user interface controls that will support at least one of your target portables), you simply use the Alias control to modify your user interface to meet the hardware requirements of another portable or text changes for a different language.

Tip: It may be easier for you to develop the base application for the target portable with the smaller screen; it is much easier to expand your user interface, rather than trying to "squeeze" it into a tight screen. If you go from a smaller interface to larger interface, you will be able to copy and paste more controls without modification. (If you design for a smaller interface first, your display texts will be shorter and you won't be required to edit your display text for the larger portable screen, whereas if you started with a large screen, you would have to make your display texts shorter for the smaller screen format).

The steps for developing an application for multiple platforms and multiple languages are essentially the same: for ease of use. However, the entire steps will be included in each section below. To introduce yourself to working with multiple platforms, please be sure to complete the levels 1,2, and 3 online tutorials (select TUTORIAL from the HELP menu).

Build Base Application

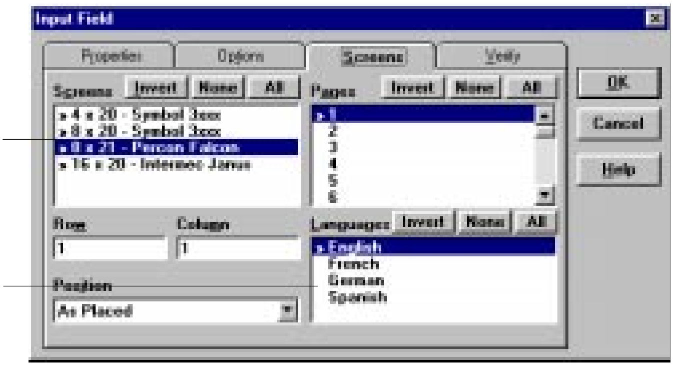
As stated earlier, building applications for multiple platforms and languages is very similar to building a single application that is the same for all portables. There is one exception: you need to make sure that when you place a user interface control on the form of your base application, that you only place the control on the primary portable screen for which the portable base application is developed (as opposed to placing the control on all portable screens). This concept will become more clear as you continue reading this chapter.

Please complete the following steps to add a user interface control to your base application's form:

1 **Verify that you are currently in the PROPERTIES tab of one of your base application's form tools.**
If you are in a different tab, please press the Properties tab to add user interface controls to your form tool.

2 **Position your cursor in the SCREEN window, right-click, and select ADD CONTROL.**
Select the user interface control that you want to add to the currently selected form tool. For more information on adding user interface controls, please see Chapter 4, Using UPG.

3 **Fill in the appropriate information for the user interface control you just added.**
Make sure that you give your control a meaningful name, define any required display text, and modify/select any other required settings. Again, please see Chapter 4, Using UPG for a more detailed explanation of each user interface control type.



Screens
➤ 4 x 20 - Symbol 3xxx
➤ 8 x 20 - Symbol 3xxx
➤ 8 x 21 - Percon Falcon
➤ 16 x 20 - Intermec Janus

Pages
➤ 1
2
3
4
5
6

Row: 1
Column: 1
Position: As Placed

Languages
➤ English
French
German
Spanish

Select the portable model screen that you want the currently selected control to appear on. This illustration shows that the currently selected input field would appear on all portable model screens.

Select the language that you want to develop your application in; English is selected by default.

Figure 7-1: Use the carets (➤) to specify where your currently selected control will appear by simply placing a caret next to each list item where you want the control to appear. For example, place a caret next to all of the portable models where you want the control to appear. Place a control next to the list item simply by clicking on the list item (clicking toggles the selection state).

4 **Press the SCREENS tab.**
The SCREENS tab allows you to control where the currently selected control will appear.

5 **Place a caret (➤) next to your primary portable in the SCREENS window.**
A portable screen is selected when a caret (➤) is placed next to the appropriate list item. Clicking on the list item toggles the current selection state. For example, if Percon Falcon was not selected, simply click on it to select it (a caret (➤) will appear). Conversely, if an item is selected, simply click on the list item to deselect it (the caret (.➤) will disappear).

If your target portable for your base application is a Percon Falcon, you will want to make sure that only the Percon Falcon is selected. If other manufacturers are selected, simply click on the portables that you want to deselect.

6 **Specify the primary language for your base application by placing a caret next to the proper list item in the LANGUAGE window.**
English is selected by default. Unless you are developing applications in foreign languages, simply leave the default selected.

7 **Press the OK button when you have completed defining the screen location for your currently selected control.**

8 **Press the PROPERTIES tab to continue adding controls.**
When you return to the Properties tab, you should see the control that you just defined appear in the screen window. If you do not see your control, press the SCREEN tab and select the appropriate target portable. If your target portable is not selected, click on the desired target portable and you will automatically be returned to the PROPERTIES tab.

Continue to build your base application in the manner just described. If you plan ahead, there may be several controls that can be shared between the primary and *secondary portables* (portables the modified base applications are built for). If you do have sharable controls, make sure to select all of the portable manufacturers that you want to use the shared control.

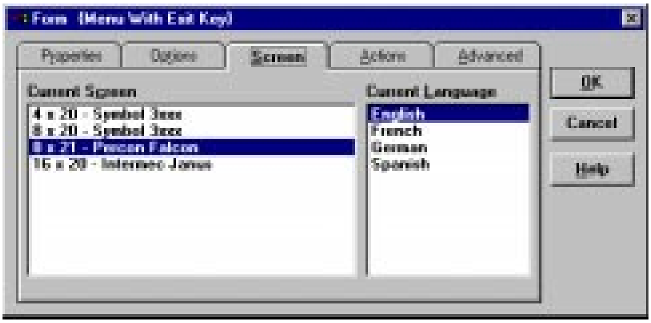
Multiple Portable Platforms

Once your base application has been built, you will need to edit each form contained in your application

The basic principle to remember when building displays for multiple portables and languages is that your currently selected control will appear according to the selections that are marked with a caret (➧). You need to make sure that the aliases you will create are only placed on the secondary portable screens.

1 **Open the form tool that you want to modify to support multiple platforms.**

2 **Press the SCREENS tab to select the secondary portable.**
After you have built your base application for the primary target portable, you need to select the next portable that you will modify your base application for.



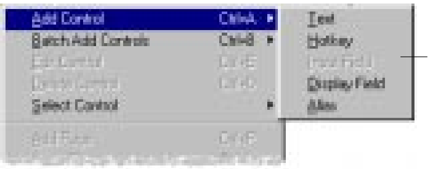
Current Screen
4 x 20 - Symbol 3xxx
8 x 20 - Symbol 3xxx
8 x 21 - Percon Falcon
16 x 20 - Intermec Janus

Current Language
English
French
German
Spanish

Figure 7-2: Select the portable and/or language that you want to define a form tool for. After you make your selection, you will automatically be returned to the Properties tab.

3 **Select the secondary portable from the CURRENT SCREEN window.**
You will automatically be returned to the Properties tab of your form tool. If you did not define any “sharable” controls when you built your base application, the Screen window in the Properties tab will be blank. (This is OK; you are about to convert this form to support multiple platforms).

4 **Position your cursor in the SCREEN window, right-click, select ADD CONTROL, select ALIAS.**
An alias allows you to place an existing control in different locations, with different display text, for different displays.



Available controls will be enabled, while unavailable controls, such as the Input Field control in this example, will be disabled.

Figure 7-3: As an alternative to right-clicking in the Screen window, you can also press <CTRL><A> to receive the pop-up menu at above right.

5 **Type the name of your alias control in the NAME field.**
This is the name that you will refer to while working within the UPG work environment.

6 **Select the existing control that you want to alias from the CONTROL TO ALIAS pull-down list.**
The functionality of the control you select from the CONTROL TO ALIAS pull-down list will be assigned to the alias you are creating. In other words, the alias control will operate as if it were the original control.

You might ask then, “If it works the same, why should I use an alias?” Using an alias will allow you to use the same control on different pages, in different locations, on different portable screens, with different display text, or any combination of these modifications. Please continue with the remaining steps, as this will become more clear.

7 **Type the text you want to display with your alias control in the DISPLAY TEXT field.**
With an alias, you can change the display text of the aliased control. This is helpful when you are converting a base application to a larger user interface. Specifically, you can specify longer, more descriptive display text for the aliased control.

8 **Press the SCREEN tab.**
You need to specify where the aliased control will appear.

9 **Place a caret (➤) in the SCREENS window next to the portable manufacturer you selected in step 3.**
Select the portable screen that you want your aliased control to appear on. Simply place a caret next to the desired portable (clicking on the list item toggles the selection status - a list item is selected when the caret is to the left of the item).

10 **Specify the location on the portable screen for your alias control.**
To staticly define the position of your control on the selected screen, select `As Placed` from the POSITION pull-down list and define the position using the ROW and COLUMN fields.

To dynamically position your control, select one of the *Bottom Row* options (If you select `Bottom Row`, the control will always appear on the bottom of your display, regardless of the display size. If you select `Bottom Row -1`, for example, your control will always appear one row above the bottom of your display.)

11 **Press the OK button when you have completed defining the screen location for your currently selected alias control.**

12 **Press the Properties tab to continue adding alias controls.**
Continue to add alias controls until you have recreated the user interface for your secondary portable(s).

To complete converting your application to support multiple hardware platforms, systematically work through your base application until you have aliased all of the necessary controls.

Multiple Languages

Once your base application has been built in the original target language, you will need to edit each form contained in your application for the secondary language.

The basic principle to remember when building displays for multiple portables and languages is that your currently selected control will appear according to the selections that are marked with a caret

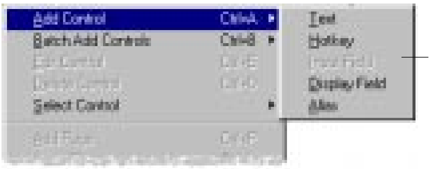
(➧). Make sure that aliased controls are placed on the secondary language screen. Translating for a foreign language involves creating aliases of the original controls and placing translated text in the Display Text field.

1 **Open the form tool that you want to modify to support multiple languages.**

2 **Press the SCREENS tab to specify the secondary development language.**
After you have built your base application for the previously selected language (most likely English), you need to select the next language that you will modify your base application for (make sure you also have the appropriate portable manufacturer(s) selected).

3 **Select the desired language from the CURRENT LANGUAGE window.**
You will automatically be returned to the Properties tab of your form tool. Unless you are working with an application that has previously defined controls for the currently selected language, the Screen window in the Properties tab will be blank. (This is OK; you are about to convert this form to support multiple languages).

4 **Position your cursor in the SCREEN window, right-click, select ADD CONTROL, select ALIAS.**
An alias allows you to place an existing control in different locations, with different display text, for different displays.



Available controls will be enabled while unavailable controls, such as the Input Field control in this example, will be disabled.

Figure 7-4: As an alternative to right-clicking in the Screen window, you can also press <CTRL><A> to receive the pop-up menu at above right.

5 **Type the name of your alias control in the NAME field.**
This is the name that you will refer to while programming within the UPG work environment.

6 **Select the existing control that you want to alias from the CONTROL TO ALIAS pull-down list.**

The functionality of the control you select from the CONTROL TO ALIAS pull-down list will be assigned to the currently selected alias. In other words, the alias control will operate as if it were the original control.

You might ask then, “If it works the same, why should I use an alias?” Using an alias will allow you to use the same control on different pages, in different locations, on different portable screens, with different display text, or any combination of these modifications. Since you are building for multiple languages, you will simply be translating the display text, but an alias will allow you to retain the functionality of the aliased control.

7 **Type the translated text you want to display with your alias control in the DISPLAY TEXT field.**

With an alias, you can change the display text of the aliased control. This is helpful when you are converting a base application to a larger user interface. Specifically, you can specify longer, more descriptive display text for the aliased control (which is often necessary during software localization).

8 **Press the SCREEN tab.**

You need to specify where the aliased control will appear.

9 **Verify a caret (➤) is placed next to the language you selected in step 3.**

Unlike specifying displays for multiple hardware platform, your alias controls for multiple languages will most likely not be shared between languages; the language you selected in step 3 will automatically be selected in the Language window.

10 **If necessary, specify the location on the portable screen for your alias control.**

To staticly define the position of your control on the selected screen, select `As Placed` from the POSITION pull-down list and define the position using the ROW and COLUMN fields.

To dynamically position your control, select one of the *Bottom Row* options (If you select `Bottom Row`, the control will always appear on the bottom of your display, regardless of the display size. If you select `Bottom Row -1`, for example, your control will always appear one row above the bottom of your display.)

11 **Press the OK button when you have completed defining the screen location for your currently selected alias control.**

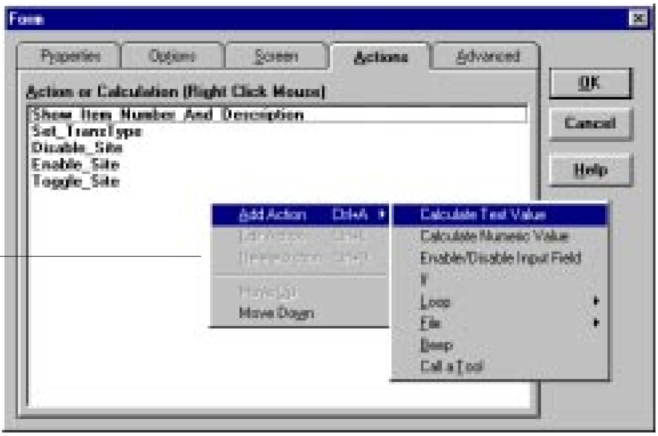
12 **Press the Properties tab to continue adding alias controls.** Continue to ad alias controls to your secondary language screens until you have translated the entire user interface.

To complete converting your application to support multiple languages, systematically work through your base application until you have aliased all of the necessary controls.

8 UPG Action Components

UPG Actions components give you the power to build custom data collection solutions by adding advanced features to your UPG generated application. The Action components allow you to perform complex calculations, establish IF-THEN-ELSE and WHILE conditional structures, and perform many file operations.

The UPG Action components are only available when you are working with tools created with the Form tooltype; press the ACTIONS tab to add Action components to your selected form.



Action or Calculation (Right Click Mouse)

Show Item Number And Description
Set_TransType
Disable_Site
Enable_Site
Toggle_Site

Position your mouse in the Action or Calculation window, right-click, and select the UPG Action component that you want to add.

Figure 8-1: The Actions tab of a tool created with the Form tooltype allows you to add UPG Action components to your application. Combining Action components allows you to add complex functionality to your UPG application.

Please complete the following steps to add a UPG Action component to your data collection application.

1 **Open the form tool that you want to add a UPG action to.**
Remember, UPG Action components can only be added to tools that were created with the form tooltype. Also, Action components that you add to this form will only be available to this form; in other words, you cannot call an Action that is not defined within the same form.

2 **Press the ACTIONS tab.**
You will assemble your action components in the ACTIONS OR CALCULATIONS window.

3 **Position your cursor in the ACTIONS OR CALCULATIONS window, right-click, and select the desired Action component.**

Please choose an Action component based on the following explanation of the common features and the individual action components. If you select a Loop or File Action component, a sub-menu will appear; select the desired Action component. For more information on using multiple Action components with each other, please see the Combining Action Components section later in this chapter.

Common Features

Like all tools and user interface controls created in UPG, each UPG Action component requires that your provide certain basic information, such as a name. The Action component also introduces several new settings, such as defining a condition/calculation and a When to Execute setting.

Name: It is especially important to enter meaningful names into the NAME field of your UPG Action components. When you use many Action components together, meaningful names will make it much easier for you to identify the purpose of each of your UPG Action components. If meaningful names are not specified, you will find yourself opening each of the Action components often to simply determine what the tool does.

You may want to use the following naming convention:

[action type]_[action name]

For example, you might enter LOOP_QtyLessThan10 in the NAME field. This will help you easily identify this Action component.

When to Execute: Use the When to Execute pull-down list to determine when your UPG Action component should be executed. See below for an explanation of all available execution times:

❑ **Setup**: The Action component is executed before the form is written to your screen. For example, if this action component is contained within a menu form, the action component will occur just before the menu appears on your portable.

- □ **After Last Input Field**: Action is executed when the user advances from the last input field of a multipage sequence (the action will be executed before you return to the loop field, if one exists). Logically, this execution time can only be used on multipage sequence forms. Your action component will not work properly if you use this execution time from within a menu or message box form.
- □ **Exit**: The Action component is executed when the form that contains the Action component is closed. For example, if you specified an Action component to be executed on exit of a sub-menu, the Action item will be executed when you close the sub-menu and return to the parent menu.
- □ **After Input Field**: The Action component will be executed after the user advances from a specified field. If AFTER INPUT FIELD is selected, another pull-down list will appear. Select the input field that you want the Action component to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list.
- □ **Called by Other Control**: The Action component is executed when it is called by a hotkey control or a Call a Tool Action component. See Using a Hotkey to execute an Action, page 4-16 for more information.

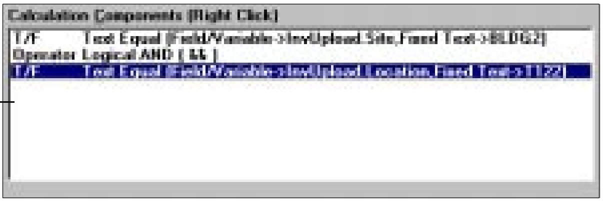
Note: The execution time selected in the When to Execute pull-down list will have precedence over the order of the UPG Actions listed in the ACTIONS OR CALCULATIONS window. For example, if you have Action01 followed by Action02 listed in the Actions or Calculations window, yet the When to Execute time for Action02 is before the When to Execute time for Action01, Action02 will occur first.

Expression Builder

Another common feature between several of the Action components is the expression builder. This feature is available when using an IF, Loop, Calculate Text Value, or Calculate Numeric Value Action components. The expression builder allows you to intuitively define condition statements and calculations without the need of writing C code.

When you are working with an IF or Loop Action component, you will be building a condition statement in the Calculation Components window. You might ask, what is a condition statement? A condition statement allows you to execute an action component(s) based on the result of the condition statement (see **Figure 8-8** for an example). For example, if you had a condition statement that checked to see if the entered value of an input field was "A123," you can execute a Beep Action component (this will produce an audible tone); if the value entered is not "A123," you can execute another action component, such as a Calculate Numeric Value Action component (or choose to have nothing happen).

You have the ability to check for multiple conditions using the Logical Operators AND, OR, or NOT (see **Figure 8-2** below). See Appendix C, Expression Builder for an explanation of each of the available Logical Operators.



Calculation Components (Right Click)

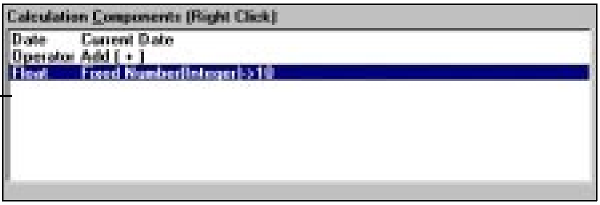
```
T/F      Text Equal (Field/Variable->InvUpload.Site,Fixed Text->BLDG2)
Operator Logical AND ( && )
T/F      Text Equal (Field/Variable->InvUpload.Location,Fixed Text->1122)
```

If both conditions do not return a value of TRUE, then the whole condition returns a value of FALSE. Use the OR logical operator to check for only one of the specified conditions.

Figure 8-2: Use a logical operator to check for multiple conditions within a single IF or Loop structure.

When you are working with the Calculate Text or Numeric Value Action component, you will be using the expression builder to construct calculations.

A calculation can be comprised of fixed values, values contained in a field, and many different mathematical operations. A calculation is constructed in a similar manner as constructing a condition. Assemble each of the calculation components in the window, making sure to include all of the necessary operators.



Calculation Components (Right Click)

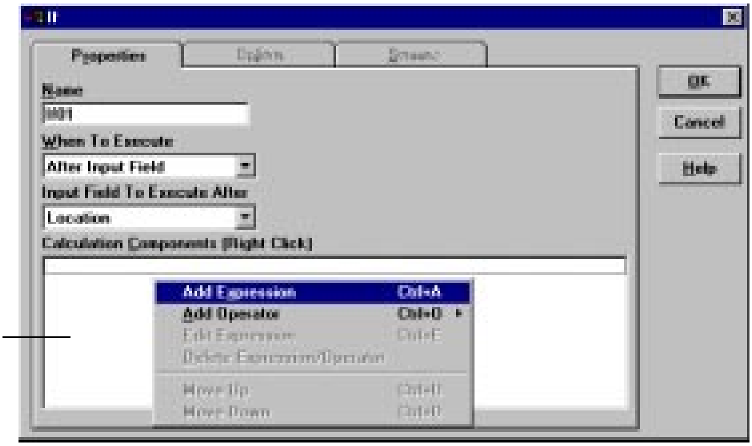
```
Date      Current Date
Operator  Add [ + ]
Float     Fixed Number(Integer)->10
```

The type of the Calculation Component is displayed to the left of the entry. The components will be executed in the order of top to bottom.

Figure 8-3: Assemble your calculation components in the Calculation window. Most compound expressions will be used with either the Calculate Text or Numeric Value Action components.

Remember, when you are building an expression within an IF or Loop Action component, you are defining a condition. If the condition is satisfied, a value of TRUE (a non-zero value) will be returned, else a value of FALSE (zero value) will be returned. You can execute another Action component based on the result of the condition.

If you are building an expression within the Calculate Text or Numeric Value Action component, you are defining a calculation; the result of the calculation will be returned.



Name: If01

When To Execute: After Input Field

Input Field To Execute After: Location

Calculation Components (Right Click)

The Calculation Components window is available when working with an IF, Loop, or one of the Calculate Value Action components.

Figure 8-4: Any conditions that you specify for the appropriate action components will appear in the Calculation Components window.

To add an expression to any of the appropriate Action components, please complete the following steps:

1 **Right-click in the CALCULATION COMPONENTS window to add an expression to your Action component.**

You can also press <CTRL><A> to add an expression to your Action component. The UPG Expression builder will appear.

2 **Select the desired expression from the Expression pull-down list.**

There are three types of expressions that can be selected.

- ❑ **Self-contained Expressions**

  Self-contained expressions do not require any more information from the UPG developer, they automatically return a value. Examples of a self-contained expression include Current Date, Current Time, and Available Disk Space.

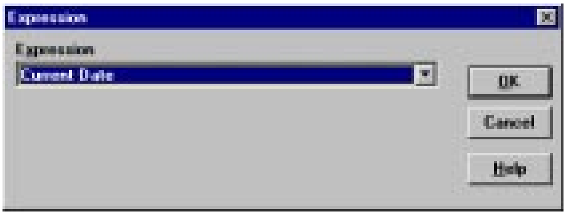
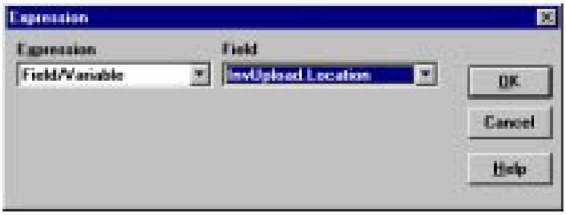


Figure 8-5: Self-contained expression do not required any more information from the UPG developer.

As you can see in **Figure 8-5**, when a self-contained expression is selected, only the EXPRESSION pull-down list is available.

- ❑ **Expressions that require specific information.**

  When an expression that requires specific information is selected, an auxiliary pull-down list or field will appear to the right of the EXPRESSION pull-down list. As the expression type suggests, you will be required to enter a fixed value or select a value/field from the auxiliary pull-down list. See Appendix C for detailed information on each of the available expressions.



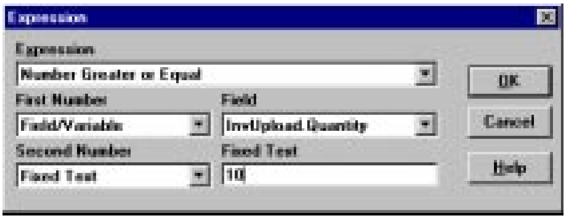
Expression: Field/Variable
Field: InvUpload Location

Figure 8-6: An expression that requires specific information will automatically place a pull-down list or entry field to the right of the expression pull-down list.

Examples of expressions that required specific information include Field/Variable or Fixed Number (Float). If you selected Field/Variable, you will need to select the field or variable from the FIELD pull-down (see **Figure 8-6**). If you selected Fixed Number (Float), you will be required to enter a Floating Point value (real number, such as `1.23`) in the FIXED field that appears.

- ❑ **Expressions that require parameters.**

  Similar to *expressions that require specific information*, you will be required to enter more information for *expressions that require parameters*. The major difference between the two types of expressions is that *expressions that require parameters* necessitate that you select two more expressions.



Expression: Number Greater or Equal
First Number: Field/Variable
Field: InvUpload Quantity
Second Number: Fixed Text
Fixed Text: 10

Figure 8-7: Expressions that require parameters allow you to easily compare values and test for equality. As you can see, depending on the expressions selected, you may define up to 5 individual pieces.

Depending on the selected parameter expressions, you may need to define some more information (when you are selecting an expression as a parameter, you can only select *self-contained* or *expressions that require specific* *information* expressions). Please see **Figure 8-7** for an example of how an expression that require parameters may look.

3 **Press the OK button to return to the UPG Action definition form when you have completed building your expression.**
You will see your newly defined expression appear in the CALCULATION COMPONENTS window.

4 **If you want to build a compound expression, press <CTRL><A> to add another expression or operator.**
Repeat steps 1-4 to continue adding components to either your condition or calculation. Make sure that you include any necessary operators.

Calculate Text Value

Use the Calculate Text Value Action component to build expressions that return a text value. Calculated values can be as simple as combining the values of two fields into one field, separated by a colon, or they can be as complex as constructing long text strings based on user entry or environment variables.

1 **Enter the name of your Calculate Text Value component in the NAME field; your name must comply with C naming conventions.**
This is the name that you will refer to while programming within the UPG environment.

2 **If you want to display the result of your Calculate Text Value component on your portable display, enable (check) the SHOW RESULT ON SCREEN checkbox.**
The WHEN TO EXECUTE field is disabled when SHOW RESULT ON SCREEN is enabled because the value will be shown on the screen (page) specified in the SCREEN tab.

If SHOW RESULT ON SCREEN is enabled, enter a label in the DISPLAY TEXT field and define the label position in the OPTIONS tab (use the FIELD POSITION pull-down list).

3 **Select the field that will store the Calculate Value result from the RESULT FIELD pull-down list; if you do not select a result field, your value will not be saved.**

4 **If SHOW RESULT ON SCREEN is disabled, select the execution time from the WHEN TO EXECUTE pull-down list.**
If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

5 **Position your mouse in the CALCULATION COMPONENTS window, right-click, and select ADD EXPRESSION.**
Please see the Expression Builder section earlier in this chapter to learn how to build calculations.

Calculate Numeric Value

Use the Calculate Numeric Value Action component to build expressions that return a numeric value. Calculated values can be as simple as adding the values of two fields together, or they can be as complex as building compound mathematical operations.

1 **Enter the name of your Calculate Numeric Value component in the NAME field; your name must comply with C naming conventions.**
This is the name that you will refer to while programming within the UPG environment.

2 **If you want to display the result of your Calculate Numeric Value component on your portable display, enable (check) the SHOW RESULT ON SCREEN checkbox.**
The WHEN TO EXECUTE field is disabled when SHOW RESULT ON SCREEN is enabled because the value will be shown on the screen (page) specified in the SCREEN tab.

If SHOW RESULT ON SCREEN is enabled, enter a label in the DISPLAY TEXT field; define the label position in the OPTIONS tab (use the FIELD POSITION pull-down list).

3 **Select the field that will store the Calculate Numeric Value result from the RESULT FIELD pull-down list. If you do not select a result field, your value will not be saved.**

4 **If SHOW RESULT ON SCREEN is disabled, select the execution time from the WHEN TO EXECUTE pull-down list.**
If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

5 **Position your mouse in the CALCULATION COMPONENTS window, right-click, and select ADD EXPRESSION.**
Please see the Expression Builder section earlier in this chapter to learn how to build calculations.

Enable/ Disable Field

The Enable/Disable action component can turn on or off a specified input field. For example, you may want to provide the user the ability to toggle an input field off an on (such as a Location input field).

1 **Enter the name of your Enable/Disable Action component in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment.

2 **Select the execution time from the WHEN TO EXECUTE pull-down list.**
If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

3 **Select which input field the Enable/Disable action component is to affect from the FIELD pull-down list.**
You will only be able to select input fields that are contained in the select multipage sequence (you cannot change the Enabled/ Disabled status of an input field contained in a multipage sequence other than the multipage sequence that contains the Enable/Disable Action component).

4 **Select the appropriate action from the ACTION pull-down list.**

- ❑ **Enable**: Target input field will be able to accept data entry.
- ❑ **Disable**: The target input field will not be able to accept data entry.
- ❑ **Toggle**: Reverses the current Enable/Disable state of the input field. For example, if an input field is currently disabled and Toggle is selected, the input field will be enabled when the Enable/Disable action component is executed.

5 **If you want to go to the field after it has been either enabled or disabled, enable (check) the GOTO FIELD AFTER CHANGE checkbox.**

Makes the target input field the active input field. This is a convenient method to go to an input field that deviates from the current multipage sequence order (allows the user to jump around in the collection sequence).

6 **Enable the MAKE NEW STATE PERSISTENT to permanently save the new Enable/Disable state.**

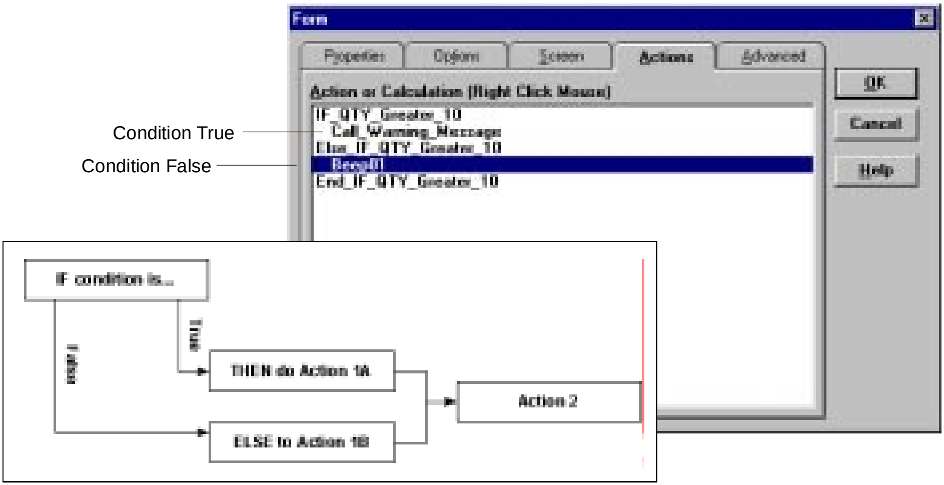
If this checkbox is enabled, the new state of the input field will persist until the status is changed by another Enable/Disable action component.

7 **Press the OK button.**

You will be returned to the ACTIONS tab of the current form; you will see your newly created Action component listed in the ACTIONS OR CALCULATIONS window.

IF-THEN-ELSE

The IF Action component allows you to build an IF-THEN-ELSE conditional structure. If the specified condition returns a value of true, the Action component after the IF line will be executed. If the condition returns a value of false, the Action component after the ELSE line will be executed.



Action or Calculation [Right Click Mouse]

```
IF_QTY_Greater_10
  Call_Warning_Message
Else_IF_QTY_Greater_10
  Beep01
End_IF_QTY_Greater_10
```

Figure 8-8: The IF-THEN-ELSE Action component creates a standard C IF structure. If the condition returns a value of true (a value of non-zero, such as 1), then the true statement will occur. If the condition returns a value of false (zero), the Else statement will occur. See a C programming reference book for more information.

1 **Enter the name of your IF Action component in the Name field.**
This is the name that you will refer to while programming within the UPG environment.

2 **If SHOW RESULT ON SCREEN is disabled, select when the condition should be checked from the WHEN TO EXECUTE pull-down list.**
If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

3 **Define the condition to be checked in the CALCULATION COMPONENTS window.**
Please see the Expression Builder section earlier in this chapter to learn how to construct a condition.

4 **Press the OK button.**
You will be returned to the Actions tab of the current form. You will notice that an empty IF-THEN-ELSE structure with the

CHECKING FOR DUPLICATE VALUES

A common feature that many data collection software developers like to include in their programs is the ability to check for duplicate values. This is a necessary feature if every entered value must be unique.

Using the Expression Builder, you can add a duplicate value check to either your IF or LOOP Action component. Based on the result of your duplicate check, you can either display a warning message or force the user to enter a new value.

Before you build your duplicate check, take note of the following:

- ❑ Name of the file that the duplicate check will performed on.
- ❑ Name of the key that the duplicate check will be performed on (the duplicate value check will be performed on all of the fields that are contained in the defined key). Please see Using Key Fields on page 4-8 for more information.

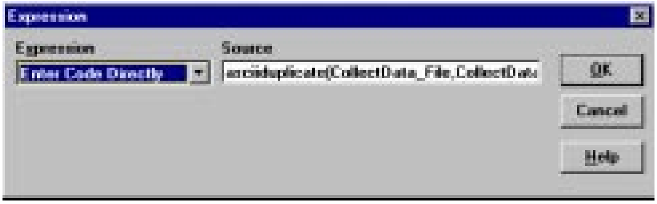
To add a duplicate check, please make sure that you are working in an IF or LOOP Action component and complete the following steps.

1 Right-click in the Calculation Components window and select Add Expression.

2 Select Enter Code Directly from the Expression pull-down list.

3 Enter the following statement into the Source field:

```
asciiduplicate([name of
file]_File,[name of file]_
[name of key])>0
```



Expression

Expression: Enter Code Directly

Source: asciiduplicate(CollectData_File,CollectDat

Where *name of file* is the name of the file that you want to check for duplicate values and *name of key* is the name of the key that contains the fields that you want to check for duplicate values (make sure you type the File and Key names exactly as they appear, including character case).

For example, you might enter:

```
asciiduplicate(CollectData
_File,CollectData_Key01)>0
```

4 Press OK to return to your IF or LOOP Action component.

If a duplicate value is found, your condition statement will return a value of TRUE. If a duplicate is not found, your condition will return a value of FALSE. Insert the actions into your IF or LOOP structure to perform the appropriate actions (such as display a warning message) based on the result of the duplicate condition check.

name you defined in step 1 is in the ACTIONS OR CALCULATIONS window.

5 **Add the Actions that occur when the condition is true.**
Make sure that the IF line of your IF-THEN-ELSE statement is highlighted, press <CTRL><A>, and select an Action component to be executed when your IF-THEN-ELSE condition statement (step #3) returns a value of true (if you want nothing to happen, skip to step #6).

Note: When you add an Action component within an IF-THEN-ELSE structure, you do not need to select a WHEN TO EXECUTE value from the pull-down list; in fact, the WHEN TO EXECUTE pull-down list will be disabled. It is not necessary to specify an execution time, because the action will be executed based on the result of IF condition statement.

6 **Add the Actions that occur when the condition is false.**
Make sure that the ELSE line of your IF-THEN-ELSE statement is highlighted, press <CTRL><A>, and select an Action component to be executed when your IF-THEN-ELSE condition statement returns a value of false (if you want nothing to happen, skip this step).

7 **Press the OK button.**
You will be returned to the Actions tab of the current form. You will see your newly created Action component listed in the ACTIONS OR CALCULATIONS window.

Loop

Use the UPG Loop action to repeat other UPG actions based on the return value of the checked condition. The Loop action component builds a standard WHILE structure. You will notice that you have several Loop Action component to choose from. Please select the required Loop Action component based on the descriptions below.

While Condition is True

The defined condition is checked at the beginning of the While loop. If the condition is true, the actions within the loop will be executed and the condition will be checked again at the top of the loop. If the condition is still true, the loop will be repeated until a false value is returned at the top of the loop (if the initial check returns a false value, the loop may never be executed).

Until Condition is True

The defined condition is checked at the beginning of the While loop. If the condition is false, the actions within the loop will be executed and the condition will be checked again at the top of the loop. If the condition is still false, the loop will be repeated until a true value is returned at the top of the loop (if the initial check returns a true value, the loop may never be executed).

For Range of Values

The For Range of Values loop creates a standard C FOR structure that uses the following syntax FOR (initialize, condition, increment). It allows you to repeat a series of steps a specified amount of times.

While and Until Condition is True Loops

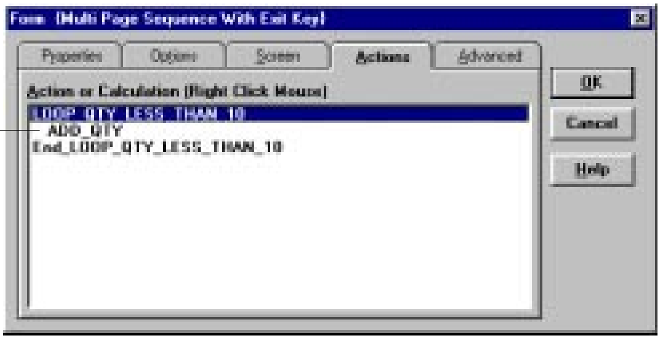
To insert a While Condition is True or an Until Condition is True Loop action component into your UPG application, complete the following steps:

1 **Enter the name of your Loop Action component in the NAME field.**
This is the name that you will refer to while programming within the UPG environment.

2 **Select a value from the WHEN TO EXECUTE pull-down list to specify when the condition should be checked.**
If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

3 **Define the condition to be checked in the CALCULATION COMPONENTS window.**
Please see the Expression Builder section earlier in this chapter to learn how to construct a condition.

4 **Press the OK button.**
You will be returned to the Actions tab of the current form. You will notice that an empty LOOP structure with the name you defined in step 1 is in the Actions or Calculations window.



Action or Calculation (Right Click Mouse)

```
LOOP_QTY_LESS_THAN_10
  ADD_QTY
End_LOOP_QTY_LESS_THAN_10
```

Insert the Action that you want repeated if the LOOP condition returns a value of *true*.

Figure 8-10: The above structure is created when using a LOOP Action component combined with another Action component inserted between the LOOP and End LOOP statements. This Action component will be repeatedly executed until the LOOP condition returns a value of *false*.

5 **Add the Action(s) that repeat when the Loop condition returns a true value.**
Make sure that the first line of your Loop structure is highlighted (i.e the `LOOP QTY LESS THAN 10` in Figure 8-10), press <CTRL><A>, and select an Action component to be executed when your LOOP condition statement (step #3) returns a value of true (if you want nothing to happen, skip to step #6).

For Range of Values Loop

A For Range of Values loop creates a standard C WHILE condition structure. Complete the following to add a For Range of Values loop to your portable application:

1 **Enter the name of your Loop Action component in the NAME field.**
This is the name that you will refer to while programming within the UPG environment.

2 **Select a value from the WHEN TO EXECUTE pull-down list to specify when the condition should be checked.**
If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

3 **Select the field that will be checked from the LOOP VARIABLE pull-down list.**
This is the value that will be checked and incremented as the loop occurs.

4 **Enter the Initialization value in the FROM field.**
This is the value that the loop will begin at; this field defaults to the value of one (1).

5 **Enter the value of the condition in the TO field.**
Once this field selected in the LOOP VARIABLE pull-down list reaches the value specified in the TO field, the loop will be terminated.

6 **Enter the Step value.**
This is the value that your loop will be incremented during each pass through the loop

7 **Press the OK button.**
You will be returned to the Actions tab of the current form. You will notice that an empty LOOP structure with the name you defined in step 1 is in the Actions or Calculations window.

8 **Add the Action(s) that repeat when the Loop condition returns a true value.**
Make sure that the first line of your Loop structure is highlighted (i.e the `LOOP QTY LESS THAN 10` in Figure 8-10), press <CTRL><A>, and select an Action component to be executed when your LOOP condition statement (step #3) returns a value of true (if you want nothing to happen, skip to step #6).

9 **Press the OK button to return to the UPG work environment.**

File

UPG File action components enable you to perform certain file and record routines, such as record navigation and record counting.

When you select File from the Actions pop-up menu, you will notice that you have several choices to select from.

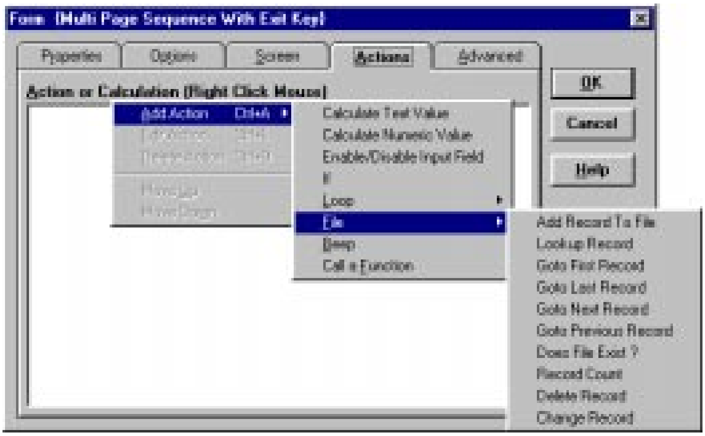


Figure 8-11: Though the File Action component has many different selections, you define the Action component is the same way: specify a name, select when the Action should execute, and select the file the Action should be performed on.

1 **Enter a name for your File Action component in the NAME field and press <TAB>.**
This is the name that you will refer to while programming within the UPG environment. Remember that your name must comply with C programming naming conventions.

2 **Select the execution time from the WHEN TO EXECUTE pull-down list.**
If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

3 **Select the target file of your action component from the FILE TO PROCESS pull-down list.**
You can only select ASCII type files (this includes file/variable tools that were created with either the ASCII or Fixed Length template). You cannot select file/variable tools created with the

INI File or Memory Variables Group template. These types of files do not contain records and are for internal application operations.

FILE ACTION COMPONENTS

Add Record to File: Writes current record to the file specified in the Properties tab of the Multipage Sequence form.

Lookup Record: This File component allows you to lookup a record based on an entered value. This component works just like the Lookup Fields in the Verify tab of an Input Field, except that you must explicitly define the Lookup Source 1.

Goto First Record: Goes to the first record contained in the file specified in the File to Process pull-down list.

Goto Last Record: Goes to the last record contained in the file specified in the File to Process pull-down list.

Goto Next Record: Goes to the next record contained in the file specified in the File to Process pull-down list.

Goto Previous Record: Goes to the previous record contained in the file specified in the File to Process pull-down list.

Does File Exist: This File component checks to see if the file specified in the File to Process pull-down lists exists.

Record Count: Counts the number of records contained in the specified file.

Delete Record: Deletes the record that is currently stored in the file buffer.

Change Record: This item is not documented (12/18/96)

Beep

Use the Beep action to produce an audible tone. (An audible tone is an important user interface element; it can be used to warn users of an error condition or to notify them than an action has occurred.) To insert a Beep Action component into your UPG application, complete the following steps:

1. **Enter the name of your Beep Action component in the NAME field.**
   This is the name that you will refer to while programming within the UPG environment.

2. **Select a value from the WHEN TO EXECUTE pull-down list to specify when the condition should be checked.**
   If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

3 **Press the OK button.**

You will be returned to the Actions tab of your current form. Your Beep Action component will be listed under the name specified in step 1.

Call a Tool

Use the Call a Tool action component to execute another tool from within a UPG action structure. This action component may be especially useful when combined with the IF or LOOP action components.

1 **Enter the name of your Call a Tool Action component in the NAME field and press <TAB>.**

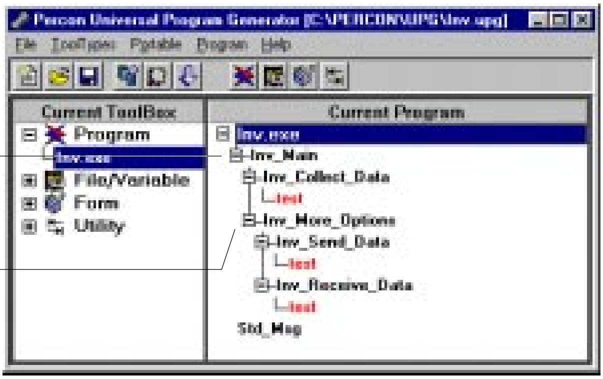
This is the name that you will refer to while programming within the UPG environment.

2 **Select the execution time from the WHEN TO EXECUTE pull-down list.**

If you select AFTER INPUT FIELD, specify the field to execute after from the INPUT FIELD TO EXECUTE AFTER pull-down list. For a detailed explanation, please see the Common Features section at the beginning of this chapter.

3 **Select the tool you want to call from the TOOL TO CALL pull-down list.**

You may select any tool in your toolbox, except those that are a parent tool. (Selecting a parent tool would lead to recursion; please see a C programming reference for more information).



In this example, Inv_Main is a parent tool to all attached tools. Subsequently, you can never select the start-up tool from the Tool to Call pull-down list.

Inv_More_Options is a parent to Inv_Send_Data and Inv_Receive_Data.

Figure 8-12: The Parent/Child relationship can be identified based on indentation and relationship lines. Any tool that is connected with a relationship line and is above and to the left of another tool is a parent.

4 **Press the OK button.**
Your Call a Tool Action will appear in the ACTIONS OR CALCULATIONS window with the name you specified earlier in step 1.

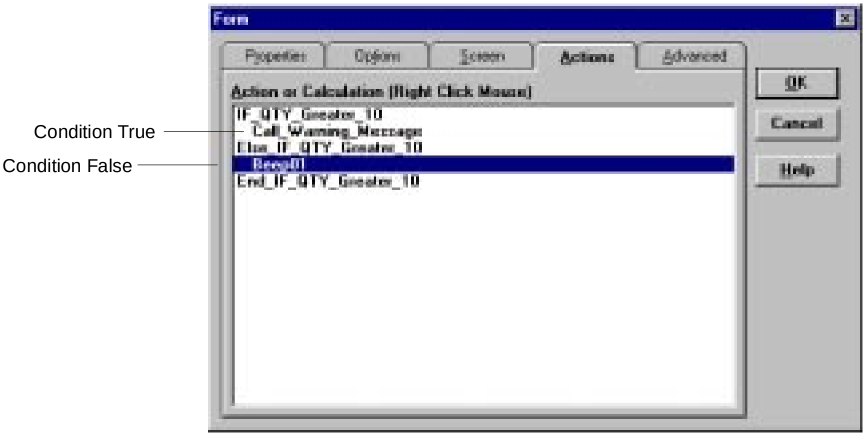
Note: When you use a Call a Tool Action component, you are essentially linking tools. When tools are called (linked) in this fashion, they are not graphically shown in the Program pane of the UPG work environment.

Combining Action Components

Using individual UPG Action components in your application can greatly increase the power of your data collection applications; combining UPG Actions, however, can create some truly sophisticated data collection routines.

As mentioned earlier in this chapter, the execution time specified in the WHEN TO EXECUTE pull-down list of each Action component takes precedence over the order of UPG Actions in the Actions or Calculations window. If, however, you have multiple UPG Actions defined and the same When to Execute value is selected, the order of execution will then be determined by the order of the Actions in the window.

You will most often combine action components when you are using an IF or Loop Action component. Based on the result of the condition, you can execute 1 or more Action components.



```
IF_QTY_Greater_10
   Call_Warning_Message
Else_IF_QTY_Greater_10
   Beep(0)
End_IF_QTY_Greater_10
```

Figure 8-13: Combining Action components, especially with the IF and Loop Action components can greatly increase the power and flexibility of your portable data collection application.

For example, if you use an IF statement to check for a specific entered value, you may want to use the Beep Action component to produce an audible tone. Specifically, if you want to display an error message if the quantity entered is greater than 10, you would use the IF Action components combined with a Call a Tool Action component. If the quantity entered is not greater than 10, an audible tone produced with the Beep action component will sound.

9 ADVANCED UPG

Customize with C Code

If you are an experienced C programmer and have specialized functions that you would like to include in your UPG generated application, you have the option of using UPG Code Hooks to insert your own C source code within the UPG generated source code.

The code you insert using UPG Code Hooks must comply with standard ANSI C programming requirements. Some C compilers are not as strict in these requirements as UPG.

Warning: Code Hooks are not for the faint of heart. Unless you are an experienced C programmer, **do not** use Code Hooks. If the code you enter is not valid C source code, you will receive compile errors. If you do receive compile errors, you will be "on your own," as the errors are a result of your source code. If you do receive compile errors, please consult your compiler documentation for an explanation. (If you are using the integrated Power C compiler, please contact MIX Software, Incorporated at (214) 783-6001 to purchase a manual; you can also email to info@mixsoftware.com.)

Considering the above, only experienced C programmers will be using the *advanced* capabilities of UPG. At this point, some clarification is needed to clearly distinguish the relationship between UPG tools and C functions.

❑ All tools created with the Form or Utility tooltype creates a standard C function. If the name of your UPG tool is InvCollect, it will create a function called InvCollect () in the UPG generated C code.

Note: To determine which UPG generated source file contains the code for a particular UPG tool, highlight the tool, right-click, and select VIEW SOURCE. When the source code appears in the code editor, the name of the UPG generated source file will be in the editor's title bar.

- ❑ All tools created with the Program tooltype create a C “main” function. Specifically, your program creates the function `void main (int argc, char *argv[]);` this will be in your `application.c` file (for example, if your application name is `inv.exe`, the main function will be in the `inv.c` file).
- ❑ All tools created with the File/Variable tooltype create a `typedef` structure in the application.h header file. For example, if your application is `inv.exe`, the header file that contains the typedef will be `invf.h`. This header file will be included in all UPG generated source files (the `Include` statement can easily be identified by looking for the `//Declarations` comment).

With the ability to add your own C code to the UPG generated C code, there may be times when you need to change the return type of a UPG function or you need to declare arguments and pass values. UPG provides the tools necessary to make these modifications.

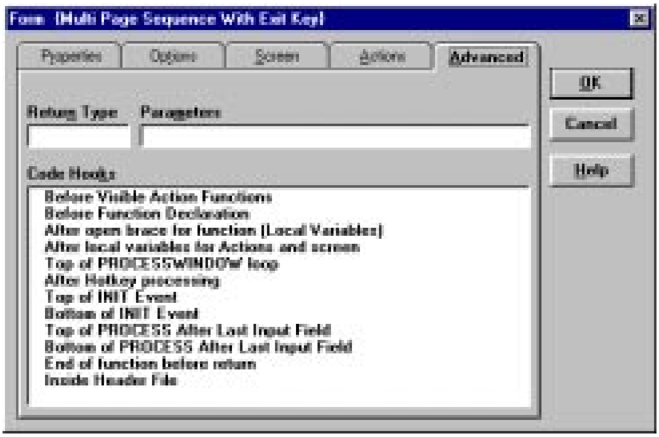
The remainder of this chapter will give you all of the necessary information needed to customize a UPG portable application with your own C code.

Embed C Code

You can insert your own C source code using UPG Code Hooks. The Code Hooks are contained in the Advanced tab of either the Program or Form tooltype definition screen.

1 **Open the tool that will contain the code hook.**
Remember, only tools created with the Program or Form tooltype can contain code hooks.

2 **Press the Advanced tab.**
A list of available code hooks will appear in the CODE HOOKS window.



Before Visible Action Functions
Before Function Declaration
After open brace for function (Local Variables)
After local variables for Actions and screen
Top of PROCESSWINDOW loop
After Hotkey processing
Top of INIT Event
Bottom of INIT Event
Top of PROCESS After Last Input Field
Bottom of PROCESS After Last Input Field
End of function before return
Inside Header File

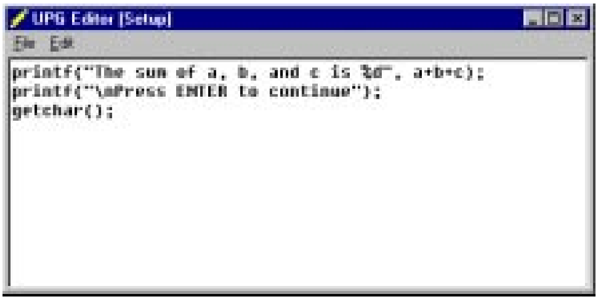
Figure 9-1: The Advanced tab of both the Program and Form definition screens are identical, except that you cannot modify the return type of a tool created with the Program tooltype, and the available code hook insertion points change with each type of template used (see the Code Hook Insertion Point tables later in this chapter.

3 **Double-click on the Code Hook insertion (embed) point where you want to place your own source code within the UPG generated source code.**

The available code hooks are insertion points within the UPG generated source code where you are able to place your own code. The code hook you select will determine what type of code you can place in this code hook. For example, if you select Before Function Declaration, you can only place declarative code within this code hook. Please see **Table 9-1** through **Table 9-4** for a listing of the different code hooks.

4 **Enter your own source code within the integrated UPG code editor.**

Be certain to begin your code at the left-most edge. Your code will be indented properly within the UPG generated source code.



```
printf("The sum of a, b, and c is %d", a+b+c);
printf("\nPress ENTER to continue");
getchar();
```

Figure 9-2: The integrated UPG code editor appears when you double-click on a code hook insertion point in the Code Hook window in a Form or Program's Advanced tab.

5 **Select SAVE from the FILE menu.**
You will be returned to the ADVANCED tab of either your program or form tool; notice that a caret (➤) is placed next to the Code Hook where your source code was inserted.

6 **Add more source code.**
If you want to enter more of your own source code into the current tool's Code Hooks, repeat steps 1-6.

7 **Press the OK button when you have completed entering your own source code.**
You will be returned to the UPG work environment. Compile your application to make sure that the source code you entered via a Code Hook is without errors. If you encounter compiler errors, the error messages and reference numbers will be displayed in the compile window. Consult your compiler documentation for further information.

The following four tables will provide you with a brief description of each code hook available in UPG. A description of the Code Hook and the type of code that can be inserted is listed for each individual Code Hook insertion point. For more information, please see Appendix B, Code Hooks. This appendix will show exactly where in the UPG generated source code the available Code Hooks are inserted.

PROGRAM Code Hook Insertion Points

Code Hook Insertion Point	Description	Valid Code
Start of Program File	Inserts your code immediately before the Main function.	Directives Global declaration
After { in Function Main	Inserts your code immediately after the opening brace ({) in the Main function.	Local Declarations Statements
Before } in Function Main	Inserts your code immediately before the closing brace (}) in the Main function.	Statements

Table 9-1

FORM (Menu Template) Code Hook Insertion Points

Code Hook Insertion Point	Description	Valid Code
Before Function Declaration	Inserts code hook immediately before the first function contained in the current form’s .C file.	Directives Global declaration
Before Screen Declaration	Code hook is inserted between the opening brace ({)of your current form’s function and the screen declaration (before the enum declaration).	local declarations
After Screen Declaration	Code hook is inserted immediately after the static declaration.	local declarations
Top of Processing Loop	Inserts code hook immediately after the opening brace ({) in the for(;;) function (following the PROCSETUP statement).	statements
Bottom of Processing Loop	Inserts code hook immediately before the closing brace (}) in the for(;;) function (following the PROCSETUP statement).	statements
Before Exit	Code hook is placed before the closing brace (}) of the PROCSETUP statement.	statements
Inside Header File	Inserts code hook at the end of you application.h header file.	global declarations

Table 9-2

FORM (Message Template) Code Hook Insertion Points		
Code Hook Insertion Point	**Description**	**Valid Code**
Before Function Declaration	Inserts code hook immediately before the first function contained in the current form’s .C file.	Directives Global declaration
Before Screen Declaration	Code hook is inserted between the opening brace ({)of your current form’s function and the screen declaration (before the enum declaration).	local declarations
After Screen Declaration	Code hook is inserted immediately after the static declaration.	local declarations
After Message Display:	Code hook is placed before the closing brace (}) of the PROCSETUP statement.	statements
Inside Header File	Inserts code hook at the end of you application.h header file.	global declarations

Table 9-3

FORM (Multipage Sequence Template) Code Hook Insertion Points		
Code Hook Insertion Point	**Description**	**Valid Code**
Before Visible Action Functions	Code hook is inserted immediately after the static declaration.	Directives Global Declarations
Before Function Declaration	Inserts code hook immediately before the first function contained in the current form’s .C file.	Directives Global declaration
After Open Brace for Function (Local Variables)	Code hook is inserted between the opening brace ({)of your current form’s function and the screen declaration (before the enum declaration).	local declarations
After Local Variable for Actions and Screen	Code hook is inserted immediately after the static declaration.	local declarations
Top of PROCESSWINDOW Loop	Inserts code hook immediately after the opening brace ({) of the PROCESSWINDOW statement.	statements
After Hotkey Processing	Code hook is inserted after the last hotkey IF statement (hotkey processing is contained within the PROCESSWINDOW structure).	statements
Top of INIT Event	Inserts code hook before the readinisettings() function.	statements
Bottom of INIT Event	Inserts code hook after the last STRUCTCLEAR statement that follows the readinisettings() function.	Statements
Top of PROCESS After Last Input Field	Code hook is inserted directly after the case -2 statement.	Statements
Bottom of PROCESS After Last Input Field	Code hook is inserted directly before the case -2, break statement.	Statements
Inside Header File	Inserts code hook at the end of you application.h header file.	global declarations

Table 9-4

Define Return Type

Since you have the ability to add your own C code to the UPG generated source code, you may insert code that will return a value. All UPG functions are type Void unless you specify otherwise in the Advanced tab of the Form definition screen (you cannot change the Return Type of a program tool).

Note: If a C function is not returning a value, then the function must be of a type Void. If a return type is defined, then the function must return a value; if a return value is not returned, you will receive compile errors.

Specifying a return type in the RETURN TYPE field requires that you include the appropriate C code to receive the returned value in the calling tool (function), either by a code hook or a compare Action (IF or Loop Action components - see Chapter 8). If you need more information on return types and returning values, please consult a C reference book or an experienced C programmer.

Please complete the following to define a Return Type:

1 **Open the tool contains the return type you want to modify.**
Remember, you can only modify the return type of tools created with the Form tooltype.

2 **Press the ADVANCED tab.**
Notice that there is a RETURN TYPE field above the listing of available Code Hooks.

3 **Enter the return type you want specified in the RETURN TYPE field.**
You must enter a valid return type in the Return Type field. If you do not enter a valid return type, you will receive compile errors.

4 **Add the code that will return a value using a Code Hook.**
Since all UPG functions do not return a value, you will need to enter the code to return a value yourself. For example, if you have a tool that is named `functioncall`, you might add the following code to return a value to the calling function (for the

following example, you will need to specify `int` as the return type):

```
ireturnval=5+15;
return ireturnval;
```

5 **Press OK to return to the UPG work environment.**
You have now modified the return type of the *Form* tool selected in step 1. You now must prepare your portable application to receive the returned value. You may either receive the returned value using a Code Hook or an Action component. Please review the appropriate steps contained in the following two sections.

Receive Value - Code Hook

1 **Open the tool that will call the target tool (function); the target tool contains the return value.**
Remember, you can only embed your own code using a Code Hook in tools created with either the Program or Form tooltypes.

2 **Press the ADVANCED tab.**
Recall from the Embed C Code section earlier in this chapter that you embed your own C code using the Code Hooks listed in the ADVANCED tab of either the Program or Form definition screens.

Note: It is important to review the Code Hook Insertion Point tables beginning on page 9-6 before embedding any of your own C code in a Code Hook. You need to make sure that you enter your code in a Code Hook that accepts statement code.

3 **Declare the variable that will receive the returned value.**
You must make your variable declarations in code hooks that allow declarations (see Embed C Code earlier in this chapter for information on using Code Hooks). You might, for example, make the following declaration:

```
int ivalue;
```

4 **Enter the code to call the target tool and receive the return value.**
You can only place the code to call the target function and receive the returned value in statement Code Hooks (non-declarative). An example of this code might look like the following:

```
ivalue=functioncall();
```

5 **When you have completed entering and saving your own code, press the OK button to return to the UPG work environment.**

Receive Value - Action Component

Using the UPG Expression Builder, you may include a returned value within the condition statement of an IF or Loop action component. For more information on using UPG Action components, please review Chapter 8 - UPG Actions Components.

1 **Open the tool that will call the target tool (function); the target tool contains the return value.**
Remember, you can only embed your own code using a Code Hook in tools created with the Form tooltype.

2 **Press the ADVANCED tab.**
Even though you are receiving a returned value with an Action component, you still need to declare the variable that will receive the returned value.

3 **Declare the variable that will receive the returned value.**
You must make your variable declarations in code hooks that allow declarations (see Embed C Code earlier in this chapter for information on using Code Hooks). You might, for example, make the following declaration:

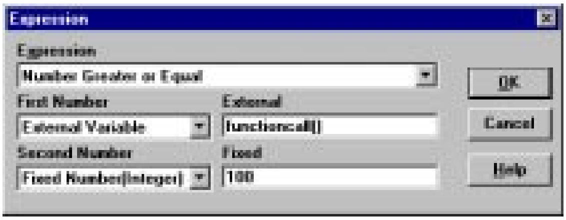
```
int ivalue;
```

4 **Press the ACTIONS tab.**

5 **Right-click in the ACTION OR CALCULATION window and select either the IF or Loop Action components.**

The IF and Loop actions components are comparative and return a value of TRUE or FALSE, based on the result of the comparison.

6 **After you have named and determined the WHEN TO EXECUTE time, right-click in the CALCULATION COMPONENT window and select ADD EXPRESSION.**



Expression

Expression: Number Greater or Equal

First Number: External Variable

External: functioncall()

Second Number: Fixed Number(Integer)

Fixed: 100

Figure 9-3: The sample expression above will check to see if the returned value from the function functioncall is greater than or equal to the fixed value of "100." If the returned value is greater than or equal to 100, the expression will return a value of TRUE; if the condition fails, a the expression will return the value of FALSE.

a **Select a logical comparison expression (i.e. Number Greater or Equal).**

b **Select EXTERNAL VARIABLE from the FIRST NUMBER pull-down list.**

See Appendix C for more information on each of the available UPG expressions.

c **Enter the call function and receive return value code in the EXTERNAL field.**

Similar to receiving a returned value with a code hook, you insert the code to receive a returned value in the External field. For example, you might enter:

```
functioncall()
```

d **Select an expression to compare the returned value against (i.e. Fixed Number (Integer) and specify 100).**

7 **Press OK when you have completed building your expression.**

8 **Press OK to return to the Form definition screen.**
You will see the Action you just defined appear in the Actions or Calculations window.

9 **Press OK to return to the UPG work environment.**

Parameter Lists

If you want to pass values between tools (C functions), you need to define the arguments that are being passed in the target function's (where the values are being passed to) parameter list. UPG provides you with the mechanism to declare these arguments; the PARAMETERS field in the ADVANCED tab allows you to declare these arguments.

Declare Arguments

1 **Open the tool that you want to add the argument declarations to.**
You may declare arguments in tools that were created with either the Program or Form tooltype.

2 **Press the ADVANCED tab.**
Notice that there is a PARAMETERS field above the listing of available Code Hooks.

3 **Declare your arguments in the Parameters field.**
You declare your arguments as if you were entering them directly in C code, except that you do not need to enclose the argument declarations within parentheses. For example, if you entered `int a, int b, int c` in the PARAMETERS field for the form `Sample`, the function would appear as `Void Sample(int a, int b, int c)` in the UPG generated source code. (Recall that all UPG functions are type Void by default; see the Define Return Type section above for more information.)

4 **Using a Code Hook, enter the code that will use the values passed to the arguments declared in step 3.**
You must enter this code into a code hook that can handle statements. Please review the Code Hook Insertion Points beginning on page 9-6 to determine which Code Hook to use.

You might, for example, enter the following code to use the values passed to the arguments:

```
printf(“The sum of a, b and c is %d”, a+b+c);
```

While there is only one way to declare these arguments, there are two method to pass values to these declared arguments. You may:

□ Pass arguments via a Code Hook

□ Pass argument references using the Call Tool Parameters when defining a Hotkey or a Call a Tool Action component.

Pass Value: Use Code Hook

You can embed your own C code to call and pass values to the tool that contains the arguments declared above. Please complete the following to call and pass values using a code hook:

1 **Open the tool that will call the tool with the declared arguments.**
You may use Code Hooks in tools that were created with either the *Program* or *Form* tooltype.

2 **Press the Advanced tab.**
All of the available Code Hooks will be listed in the CODE HOOK window.

Note: It is important to review the Code Hook Insertion Point tables beginning on page 9-6 before embedding any of your own C code in a Code Hook. You need to make sure that you enter your code in a Code Hook that accepts statement code.

---

3 **Enter the code that will call the tool with the declared arguments and pass the desired values.**
You must enter this code into a code hook that can handle statements. Please review the Code Hook Insertion Points beginning on page 9-6 to determine which Code Hook to use.

Continuing with the examples established in the previous section, you might enter the following code into a Setup code hook:

```
Sample(4, 4, 4);
```

4 **Press OK to return to the UPG work environment.**
You will notice that the tool that you referenced in the Code Hook will not appear in the Program pane (unless that particular tool is called from another location within your portable application).

5 **Highlight the tool with the declared arguments and select ADD UNATTACHED FUNCTION from the PROGRAM→ ADVANCED menu.**
Since the tool with the declared arguments is being called from within a Code Hook, the tool will not automatically be placed within the program pane, and therefore will not be compiled. You must use the ADD UNATTACHED FUNCTION menu selection to add the tool with the declared arguments to your portable application. You should now see the tool with the declared arguments at the bottom of your Program pane, below the rest of your portable application structure.

Pass Value: Use Call Tool Parameters

You can also pass parameters when you use a Hotkey or Call Tool Action component to call another tool. If you use this method, the tool that contains the declared arguments will automatically be linked to your existing program structure, and you will not be required to manually add the tool with the declared arguments to your program (unlike using a Code Hook to call the tool - see above).

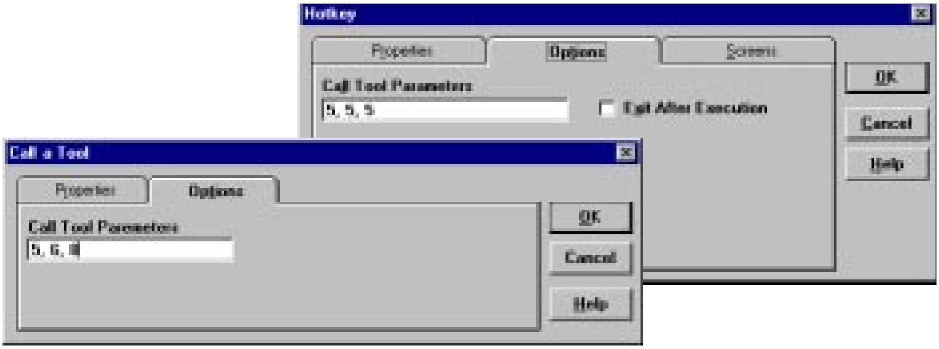
1 **Open the tool that will call the tool with the declared arguments.**
You can only use tools created with the Form tooltype to pass parameters using the CALL TOOL PARAMETERS field contained in the Hotkey or Call a Tool Action OPTIONS tab.

2 **Attach the tool with the declared values using a Hotkey (see Chapter 4 for information on using a Hotkey) or a Call**

a Tool Action component (see Chapter 8 for information on the Call a Tool Action component).

3 **Open the Options tab of the Hotkey or Call a Tool Action component used to call the tool with the declared arguments.**

You will see a CALL TOOL PARAMETERS field in the OPTIONS tab.



Call Tool Parameters: 5, 5, 5

Call Tool Parameters: 5, 6, 8

Figure 9-4: The values entered into the Call Tool Parameters field will be passed to the arguments declared in the tool being called.

4 **Enter the values to be passed to the called tool with the declared arguments in the CALL TOOL PARAMETERS field.**

Enter the values as you would in C code, but without the parenthesis. For example, you might enter 3, 4, 5 in the CALL TOOL PARAMETERS field.

For example, if you called the Sample tool using a Hotkey or Call a Tool Action component, the following code will be generated to call the tool with the declared arguments:

```
Sample(3, 4, 5);
```

You will notice that the generated source code is the same code that was entered into the code hook in step 3 in the Pass Value - Code Hook section above.

Note: If you want to pass the value of field defined in your UPG file structure, enter the `file.field` combination as the value passed to the tool with the declared arguments. For example, to send the value of the field `Qty` contained in the `Collect` file, enter `Collect.Qty` as the value to be passed; whatever value that is contained in the file buffer for that field will be passed to the tool called.

Attaching External Files

With the ability to add your own code within the UPG generated source code, a situation may arise where you need to attach an external library, source file, or header file to your application. Attaching these external files can greatly reduce the time it takes to create a custom application, since the functions, declarations, and prototypes in their respective files can be continuously reused.

Libraries and Source Files

Attaching external libraries and source files will provide you with the ability to link your pre-existing libraries and source files to your UPG generated application. Linking these external files will allow you to call the functions in your pre-existing libraries and source files from within a UPG Code Hook. When you link an external library or source file to your UPG portable application, you also need to attach the corresponding header file (please see the following section, Header Files).

Note: While you can attach external libraries and source files to your UPG application, you can only call a function contained within these external files from a Code Hook.

1. **Select ADVANCED from the PROGRAM menu.**
2. **Select ADD EXTERNAL LIBRARY or SOURCE FILE from the ADVANCED sub-menu.**
   A standard Windows Browse dialog box will appear.
3. **Highlight the file you want to attach.**
   If necessary, navigate to the appropriate directory that contains the library or source file that you want to attach to your application.

4 **Press the SAVE button.**
You will notice that the file (including the fully qualified path) will be displayed at the bottom of the Program pane, below your portable application structure. It will be labeled with a `Lib` prefix.

Once the proper library or libraries have been installed, make sure that you make the appropriate calls from a Code Hook.

Header Files

When you attach external libraries and source files (see previous section), you also need to attach the corresponding header files. Attaching an external header file will make all of the necessary include declarations in the UPG generated source files; this will make the functions in your external libraries and source files accessible to all UPG Code Hooks. If you need more information on working with libraries, source files, and header files, please consult a C reference book or an experienced C programmer.

1 **Select ADVANCED from the PROGRAM menu.**

2 **Select ADD EXTERNAL HEADER FILE from the ADVANCED sub-menu.**
A standard Windows Browse dialog box will appear.

3 **Highlight the header file you want to attach.**
If necessary, navigate to the appropriate directory that contains the library or source file that you want to attach to your application.

4 **Press the SAVE button.**

You will notice that the header file (including the fully qualified path) will be displayed at the bottom of the Program pane, below your portable application structure; it will be labeled with a `Header` prefix.

10 UPG Communications

UPG

Communications Overview

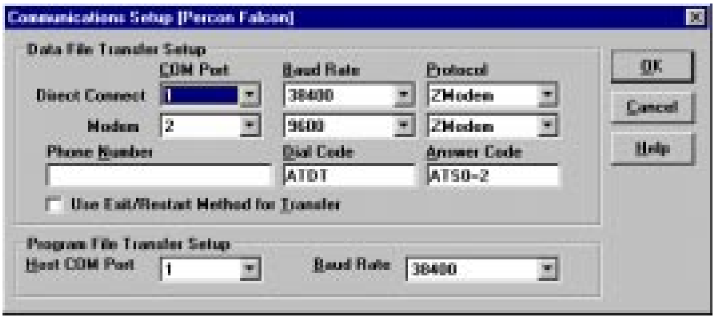
The integrated UPG communications utilities have been developed to support the most common of file transfers. You may build UPG portable applications that send and receive files with either a direct connection or modem connection between the portable and host PC. The UPG integrated communications utilities support the XModem and ZModem protocols.

COM Setup

Before you can begin communicating between your target portable(s) and your host PC, you need to setup your communications settings.

1 **Select the target portable from the PORTABLE menu that you want to define communication settings for.**
If you are building applications for multiple portables, you need to individually define the communications settings for each portable. When you have completed defining the communication settings for one portable, you need to select the remaining target portables and repeat the following steps for each portable.

2 **Select COMMUNICATIONS SETUP from the PORTABLE menu.**
All of the communication settings for your portable are contained in this window.



Communications Setup [Percon Falcon]

Data File Transfer Setup

	COM Port	Baud Rate	Protocol
Direct Connect	1	38400	ZModem
Modem	2	9600	ZModem

Phone Number:

Dial Code: ATDT

Answer Code: ATS0=2

□ Use Exit/Restart Method for Transfer

Program File Transfer Setup

Host COM Port: 1

Baud Rate: 38400

Figure 10-1: The currently selected portable will appear in the title bar of the Communication Settings window to remind you which portable is currently selected.

3 **You will notice that the COMMUNICATIONS SETUP window is divided into 2 categories: DATA FILE TRANSFER SETUP and PROGRAM FILE TRANSFER SETUP.**
It is often necessary to specify a different baud rate between data file transfers and program file transfers. Please configure your communications settings according to the following two sections.

Data File Transfer Setup

You need to define the proper settings for your portable when communicating via a direct connection or a modem.

1 **Select the proper COM port for both direct and modem connections from their respective COM PORT pull-down lists.**

Note: If you will only be using a direct connection, it is not necessary to modify any of the modem settings, including Phone Number and both the Answer and Dial Codes.

2 **Select the proper baud rate for both direct and modem connections from their respective BAUD RATE pull-down lists.**
Remember, modem connections are usually much slower than a direct connection. Achievable modem baud rates may be much lower than the rate you select due to poor telephone line conditions (static or “noise” will cause your modem to lower the baud rate to compensate for poor line conditions).

3 **Select the protocol to be used by both a direct and modem connection from their respective PROTOCOL pull-down lists.**
If you will be transferring data between the portable and the host PC using the integrated File Transfer Manager or the UPG Runtime File Transfer Manager, it is suggested that you leave ZModem as the default setting (ZModem typically has better error correction and faster file transfer times).

4 **If your UPG portable application will be using a modem, please also complete the following configuration steps:**

If you are not developing an application that will use modem communications, you can skip ahead to the Program File Transfer Setup section.

a **Enter the default phone number your portable will be dialing to send files to a host PC in the PHONE NUMBER field.**

Your users will also have the ability to change the target phone number when they execute a function you built with the Send File to Host *Utility* template.

b **Modify the DIAL CODE if you require special AT commands to send a file(s) via a modem.**

You will generally not have to edit the value in the DIAL CODE field; ATDT is the standard dial command.

c **Modify the ANSWER CODE if your require special AT commands to receive a file(s) via a modem.**

Again, you will generally not have to edit the value in the ANSWER CODE field. You may, however, wish to modify the number of rings that will pass before your modem answers. The default is 2; simply change the numerical value in the Answer Code to the number of rings you want to pass before your modem answers.

Program File Transfer Setup

As you can see, you only need to modify the baud rate at which your portable will be programmed and the COM port your PC will use to communicate with your portable. When you open the Communications Setup window, the default Program baud rate will be set to the optimum program rate for the currently selected portable. The Host COM Port defaults to COM 1. If necessary, please complete the following steps:

1 **Select the COM port that your portable data collection unit is attached to on your PC from the Host COM Port pull-down list.**

This is the COM port that your PC will use to communicate with your portable

2 **Select the baud rate that will be used when programming the currently selected portable from the BAUD RATE pull-down list in the PROGRAM FILE TRANSFER SETUP window.**
Consult your portable documentation for supported baud rates.

3 **Press the OK button to close the COMMUNICATIONS SETUP window.**
Remember, if you are developing for multiple portables, you need to select the remaining portables and repeat the COM Setup section for each portable.

Programming Portable

UPG has automated the portable programming process for you. Since different manufacturers are supported, several different programming methods are required. UPG automatically performs the proper programming routine based on the portable selected in the Portable menu.

Please complete the following steps to program your portable data collection unit:

1 **Make sure that your UPG portable application has been compiled.**
If you need more information on compiling applications, please see the Compile/Test Project section on page 4-37.

2 **Verify that your portable is properly attached to your host PC.**
If your portable uses a dock, make sure that the dock is properly attached to your PC. Many docks require that you use a null-modem with a serial cable and power supply; check your portable/dock documentation for more information.

3 **Select SEND PROGRAM TO PORTABLE from the PORTABLE menu.**

Since each portable requires a different programming procedure, please follow the on-screen instructions to complete programming your portable.

Sending and Receiving Data

UPG uses the File Transfer Manger to help you transfer data (collection, validation, and INI files) between a portable data collection unit and your host PC. You can send and receive files with the File Transfer Manager without having to switch between different utilities.

Note: The File Transfer Manager only works with direct connections between a host PC and portable data collection unit. If you are trying to receive data from a portable via modem, please use a communications utility that supports a modem. Use the same protocol as selected in the Communication Settings window to receive a file via modem using the external communications package; please see your communication package's documentation for more information.

Preparing to Transfer Files

1 **Verify that your portable is properly attached to your host PC.**
If your portable uses a dock, make sure that the dock is properly attached to your PC. Many docks require that you use a null-modem with a serial cable and power supply. Check your portable/dock documentation for more information.

2 **Select FILE TRANSFER MANAGER from the PORTABLE menu.**
If you have not configured your communication settings as detailed in the COM Setup section earlier in this chapter, the Communications Setup window will automatically appear. You need to configure your communication settings before you can use the File Transfer Manager. Please review the steps in the COM Setup section to configure your communication settings.

Select the file or files that you want to send to the target portable; press <CTRL> as you highlight the files to select more than one file.

Open the folder that contains the files you want to send to your portable or where you want to save received files to.

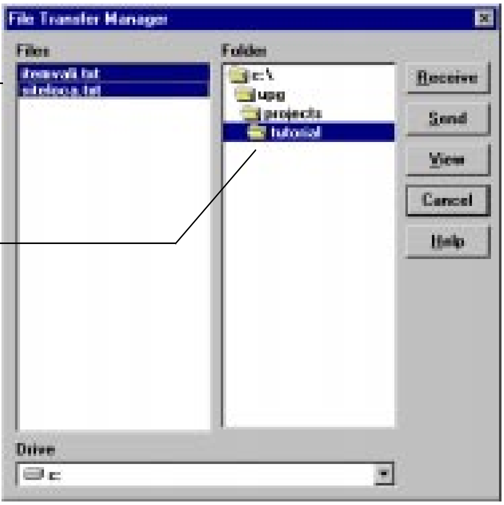


Figure 10-2: The UPG File Transfer Manager can handle all direct (with serial cable or portable dock) file transfer sessions. Use the File Transfer Manager to receive data collection files from your portable or to send validation and/or customized INI files to your portable.

Send File to Portable

1 **Highlight the file that you want to send to your portable.**
Make sure that you select a file that your portable is expecting to receive

If your portable application has a Transfer File Host Mode tool defined, you can also send multiple files to your portable. To select a series of files to send, hold the <CTRL> button as you select the files.

2 **Activate the receive utility on your portable.**
If you have built an application that supports both a direct and modem connect, make sure that you specify a direct connection for this transfer session.

If you want to receive several files, make sure that you activate a File Transfer Host Mode utility.

Note: See page 4-31 to learn how to build a Receive File From Host utility or see page 4-29 to learn how to build a Transfer File Host Mode utility.

3 **Press the Send button to begin your file transfer.**
You will see the progress of your file transfer on both your host PC and on your portable. The progress will be represented as a percent (%) complete.

4 **When the file transfer is complete (including a transfer of multiple files), your host PC will say "`Transfer Complete, Press any key to continue...`"**
When you press a key, you will be returned to the UPG File Transfer Manager.

If you want to perform another file transfer session, simply repeat the above steps. If you have finished all file transfer sessions, press the CANCEL button to return to the UPG work environment.

Receive Files from Portable

If the UPG File Transfer Manager is not open, please complete the steps listed in the Preparing to Transfer Files section.

1 **Open the directory in the FOLDER window where you want to save the file you will receive from the portable.**
Double-click on the folder to open it. If you need to change drives, select the appropriate drive from the DRIVE pull-down list.

2 **Press the RECEIVE button.**
Your PC will be placed in host receive mode. This means that you will continue receiving files until you terminate the transfer session.

3 **Begin your send function on your portable.**
If you have built an application that supports both a direct and modem connect, make sure that you specify a direct connection for this transfer session.

When the file has completed being sent to the host PC, your portable application will return to the menu where the send file function was activated.

Note: See page 4-33 to learn how to build a Send File To Host utility.

Repeat this step until you have complete sending all of the files from your portable to the host PC

4 **When you have completed receiving files, press <ESC> on your host PC.**
You will be returned to the UPG File Transfer Manager.

If you want to perform another file transfer session, simply repeat the above steps. If you have finished all file transfer sessions, press the CANCEL button to return to the UPG work environment.

View a File

At times, you may want to view a file before sending it to a portable, or you may want to view a file after you have received from the portable. You will notice that there is a View button on the File Transfer Manager. Complete the following to view a file:

1 **Highlight the file in the Files window that you want to view.**

2 **Press the View button.**
Your file will be displayed in the Source Editor (Windows Notepad by default) defined in the Options window (see page 2-6 for more information on defining a Source Editor).

3 **When you have completed viewing the file, close your Source Editor.**

Press the CANCEL button to close the File Transfer Manager and return to the UPG work environment.

Distributing your Application

If you are developing portable applications for multiple sites or wide-scale distribution, chances are that you are not going to personally program every portable that will use your portable application. It is then necessary to have a method to transfer your portable application to your users with all of the appropriate files. You also will want your portable users to be able to easily send and receive data files (such as collected data and verification files).

UPG contains a feature that allows you to easily make distribution disks (select MAKE RUNTIME DISK from the PROGRAM menu) for your portable application. A distribution disk will contain all of the necessary files for UPG Runtime to program a target portable.

Percon offers UPG Runtime, available separately from your authorized Percon reseller (part number 00-736-10). UPG Runtime allows users who don’t have the Universal Program Generator development system to program their portable and/or send or receive data with a UPG generated application.

UPG Runtime allows users to:

❑ Program Portable

❑ Send Data to Portable

❑ Receive Data from Portable

❑ Use XModem or ZModem for Data Transfer

Note: A UPG Runtime license is required for each Host PC that will communicate with a portable. Contact your Percon reseller for details.

After the runtime disk has been made, simply give it to the user with UPG Runtime and they will be able to easily install your UPG generated application on their portable, as well as transfer data between their portable and host PC.

To make a runtime disk, complete the following:

Note: Before you make your runtime disk, make sure you have selected the correct portable from the Portable menu and have compiled the application. If you don’t compile, the correct files may not be copied to your runtime disk.

1 **Select MAKE RUNTIME DISK from the PORTABLE menu.**

2 **Select the drive that you want to save the distribution files to from the DRIVE pull-down list.**
This can either be a floppy disk or a network drive.

3 **If necessary, select a directory where you want to save the runtime files to in the FOLDERS window.**
If you are saving the files to a floppy disk, you will usually want to select the root directory.

4 **Press the OK button.**

A

APPENDIX - CODE HOOKS

PROGRAM Code Hook Insertion Points

```
/*****************************************************************************************
* Percon Universal Program Generator Module                                              *
* Program   : c:\percon\upg\projects\itrak\Inv                                           *
* Dictionary: c:\percon\upg\projects\itrak\itrak.upg                                     *
*                                                                                        *
* CAUTION!!! CHANGES MADE TO THIS FILE WILL BE OVERWRITTEN ON NEXT GENERATE              *
*****************************************************************************************/

#define MAIN                                           // Put global variables in this module
#include "upg.h"                                           // UPG Global Declarations
#include "c:\percon\upg\projects\itrak\Inv.h"              // Declarations
#include "c:\percon\upg\projects\itrak\Inv.p"              // Prototypes
#include "c:\percon\upg\projects\itrak\Invf.h"             // File & Variable Declarations

#if __TURBOC__
extern unsigned _stklen  = 8*1024;                         // Set Stack  & Heap Size for
Borland Compiler
extern unsigned _heaplen = 6*1024;
#endif
SCREENSIZE gzScreensize[] = {{0,0},{19,3},{19,7},{20,7},{19,15},{0,0}};

//---> Begin Hook:Start of Program File
//-->Start of Program File code hook insertion point
//---> End Hook
void main (int argc, char *argv[])                         // Program Entry Point
{
   //---> Begin Hook:After { in Function main
   //-->After { in Function main
   //---> End Hook
   upginit(argc, argv, False);                             // Set up the UPG Library
   openall();                                              // Open All Data and Lookup Files
   Inv_Main();                                             // Call First Function in program
   closeall();                                           // Close All Data and Lookup Files"
   //---> Begin Hook:Before } in Function main
   //-->Before } in Function main
   //---> End Hook
}

/*****************************************************************************************
* The following functions are used in the program but aren’t defined yet                 *
*****************************************************************************************/

void Inv_Find(void)                {Std_Msg(1);}           // Function not defined  yet
```

FORM (Menu) Code Hook Insertion Points

```
/*****************************************************************************************
* Percon Universal Program Generator Module                                              *
* Program   : C:\PERCON\UPG\PROJECTS\ITRAK\Inv                                           *
* Dictionary: C:\PERCON\UPG\PROJECTS\ITRAK\itrak.upg                                     *
*                                                                                        *
* CAUTION!!! CHANGES MADE TO THIS FILE WILL BE OVERWRITTEN ON NEXT GENERATE              *
*****************************************************************************************/

#include "upg.h"                                           // UPG Global Declarations
#include "C:\PERCON\UPG\PROJECTS\ITRAK\Inv.h"              // Declarations
#include "C:\PERCON\UPG\PROJECTS\ITRAK\Inv.p"              // Prototypes
#include "C:\PERCON\UPG\PROJECTS\ITRAK\Invf.h"             // File & Variable Declarations

//---> Begin Hook:Before Function Declaration
//-->Before Function Declaration code hook insertion point
//---> End Hook
void Inv_Main(void)
{
   //---> Begin Hook:Before Screen Declaration
   //-->Before Screen Declaration code hook insertion point
   //---> End Hook
   enum {S_Title,S_A_Title01,S_Text02,S_Hotkey01,S_Physical_Inv,S_A_Physical_Inv01,
         S_Hotkey05,S_Move_Inv,S_A_Move_Inv01,S_Hotkey06,S_More_Options,S_A_More_Options01,
         S_Hotkey07,S_Hotkey08};
   static SCREENOBJ azScreenobj[] =
   {
      {CTRL_TEXT,"Title",-1,1,0,0,0,0,0,0,NULL,"*****Inventory*******"},
      {CTRL_ALIAS,"A_Title01",-1,2,0,0,0,0,0,S_Title,NULL,"***le Inventory*****"},
      {CTRL_TEXT,"Text02",-1,4,0,0,0,0,0,0,NULL,"***Guten Inventory****"},
      {CTRL_HOTKEY,"Hotkey01",-1,1,0,1,0,0,0,F2,NULL,"F2=Physical Inv"},
      {CTRL_HOTKEY,"Physical_Inv",-1,1,0,1,0,0,0,F2,NULL,"F2=Physical Inv"},
      {CTRL_ALIAS,"A_Physical_Inv01",-1,2,0,1,0,0,0,S_Physical_Inv,NULL,"F2=le Physical
Inv"},
      {CTRL_HOTKEY,"Hotkey05",-1,4,0,1,0,0,0,F2,NULL,"F2=Guten Physical"},
      {CTRL_HOTKEY,"Move_Inv",-1,1,0,2,0,0,0,F3,NULL,"F3=Move Inventory"},
      {CTRL_ALIAS,"A_Move_Inv01",-1,2,0,2,0,0,0,S_Move_Inv,NULL,"F3=le Move Inventory"},
      {CTRL_HOTKEY,"Hotkey06",-1,4,0,2,0,0,0,F3,NULL,"F3=Guten Move"},
      {CTRL_HOTKEY,"More_Options",-1,1,0,3,0,0,0,F4,NULL,"F4=More Options"},
      {CTRL_ALIAS,"A_More_Options01",-1,2,0,3,0,0,0,S_More_Options,NULL,"F4=le More Op-
tions"},
      {CTRL_HOTKEY,"Hotkey07",-1,4,0,3,0,0,0,F4,NULL,"F4=Guten More"},
      {CTRL_HOTKEY,"Hotkey08",-1,1,0,4,0,0,0,F5,NULL,"F5=Glibnork"},
      {0}                                                // Null struct to mark end of array
   };
   //---> Begin Hook:After Screen Declaration
   //-->After Screen Declaration code hook insertion point
   //---> End Hook
   PROCSETUP
   {
      for(;;)                                             // Loop forever (until user exits
with exit key)
      {
         //---> Begin Hook:Top of Processing Loop
         //-->Top of Processing Loop code hook insertion point
         //---> End Hook
         if(upgmenu(WIND_CLS,-1) == UPGEXITKEY)
            break;
         if(upghotkey(S_Hotkey01))                         // Hotkey {F2}
         {
            Inv_Physical_Inv();
            continue;
         }

         if(upghotkey(S_Physical_Inv))                     // Hotkey {F2}
         {
            Inv_Physical_Inv();
            continue;
         }

         if(upghotkey(S_Hotkey05))                         // Hotkey {F2}
         {
            Inv_Physical_Inv();
            continue;
         }

         if(upghotkey(S_Move_Inv))                         // Hotkey {F3}
         {
            Inv_Move_Inventory();
            continue;
         }

         if(upghotkey(S_Hotkey06))                         // Hotkey {F3}
         {
            Inv_Move_Inventory();
            continue;
         }

         if(upghotkey(S_More_Options))                     // Hotkey {F4}
         {
            Inv_More_Options();
            continue;
         }

         if(upghotkey(S_Hotkey07))                         // Hotkey {F4}
         {
            Inv_More_Options();
            continue;
         }

         if(upghotkey(S_Hotkey08))                         // Hotkey {F5}
         {
            Inv_Glibnork();
            continue;
         }
         //---> Begin Hook:Bottom of Processing Loop
         //-->Bottom of Processing Loop code hook insertion point
         //---> End Hook
      }

      //---> Begin Hook:Before Exit
      //-->Before Exit code hook insertion point
      //---> End Hook
   }
   PROCSHUTDOWN
}
```

FORM (Message) Code Hook Insertion Points

```
/*****************************************************************************************
* Percon Universal Program Generator Module                                              *
* Program   : c:\percon\upg\projects\itrak\Inv                                           *
* Dictionary: c:\percon\upg\projects\itrak\itrak.upg                                     *
*                                                                                        *
* CAUTION!!! CHANGES MADE TO THIS FILE WILL BE OVERWRITTEN ON NEXT GENERATE              *
*****************************************************************************************/

#include "upg.h"                                           // UPG Global Declarations
#include "c:\percon\upg\projects\itrak\Inv.h"              // Declarations
#include "c:\percon\upg\projects\itrak\Inv.p"              // Prototypes
#include "c:\percon\upg\projects\itrak\Invf.h"             // File & Variable Declarations

//---> Begin Hook:Before Function Declaration
//-->Before Function Declaration code hook insertion point
//---> End Hook
void MsgBox_Zero(void)
{
   //---> Begin Hook:Before Screen Declaration
   //-->Before Screen Declaration code hook insertion point
   //---> End Hook
   enum {S_Std_Message_AnyKey,S_Text01,S_Text02};
   static SCREENOBJ azScreenobj[] =
   {
      {CTRL_TEXT,"Std_Message_AnyKey",-1,1,0,-1,1,0,0,0,NULL,"Any Key To Continue"},
      {CTRL_TEXT,"Text01",-1,1,0,0,0,0,0,0,NULL,"You have entered a"},
      {CTRL_TEXT,"Text02",-1,1,0,1,0,0,0,0,NULL,"quantity of zero (0)."},
      {0}                                                  // Null struct to mark end of array
   };
   //---> Begin Hook:After Screen Declaration
   //-->After Screen Declaration code hook insertion point
   //---> End Hook
   PROCSETUP
   {
      upgmenu(WIND_CLS,-1);
      //---> Begin Hook:After Message Display
      //-->After Message Display code hook insertion point
      //---> End Hook
   }
   PROCSHUTDOWN
}
```

FORM (Multipage Sequence) Code Hook Insertion Points

```
/*****************************************************************************************
* Percon Universal Program Generator Module                                              *
* Program   : E:\PERCON\UPG\PROJECTS\ITRAK\Inv                                           *
* Dictionary: E:\PERCON\UPG\PROJECTS\ITRAK\itrak.upg                                     *
*                                                                                        *
* CAUTION!!! CHANGES MADE TO THIS FILE WILL BE OVERWRITTEN ON NEXT GENERATE              *
*****************************************************************************************/

#include "upg.h"                                           // UPG Global Declarations
#include "E:\PERCON\UPG\PROJECTS\ITRAK\Inv.h"              // Declarations
#include "E:\PERCON\UPG\PROJECTS\ITRAK\Inv.p"              // Prototypes
#include "E:\PERCON\UPG\PROJECTS\ITRAK\Invf.h"             // File & Variable Declarations

//---> Begin Hook:Before Visible Action Functions
//-->Before Visible Action Functions code hook insertion point
//---> End Hook
static void Calc_Show_ItemNum_and_Description(void)
{
   sprintf(gsCalcresult,"%s/%s",InvUpload.Item_Number,Item.Description);  // Show_ItemNum_
and_Description
}

//---> Begin Hook:Before Function Declaration
//-->Before Function Declaration code hook insertion point
//---> End Hook
void Inv_Move_Inventory(void)
{
   //---> Begin Hook:After open brace for function (Local Variables)
   //-->After Open Brace for Function (Local Variables) code hook insertion point
   //---> End Hook
   enum {S_User_ID,S_Site,S_Item_Number,S_Lot_Number,S_Exp_Date,S_Move_From,S_Move_To_Site,
         S_Move_To,S_Quantity,S_Std_Hotkey_Exit,S_User_ID_A01,S_Site_A01,S_Hotkey01,
         S_Show_ItemNum_and_Description};
   static SCREENOBJ azScreenobj[] =
   {
      {CTRL_ENTRY,"User_ID",-1,1,0,0,0,1,1,0,NULL,"User ID",
         PROP_TEXT|PROP_REQ,25,&InvUpload.User_ID},
      {CTRL_ENTRY,"Site",-1,1,0,0,0,2,1,0,NULL,"Site",
         PROP_TEXT|PROP_REQ,25,&InvUpload.Site,"(none)"},
      {CTRL_ENTRY,"Item_Number",-1,1,0,0,0,4,1,0,NULL,"Item Number",
         PROP_TEXT|PROP_REQ,25,&InvUpload.Item_Number},
      {CTRL_ENTRY,"Lot_Number",-1,1,0,1,0,8,1,0,NULL,"Lot Number",
         PROP_TEXT|PROP_REQ,25,&InvUpload.Lot_Number},
      {CTRL_ENTRY,"Exp_Date",-1,1,0,1,0,16,1,0,NULL,"Expiration Date",
         PROP_DATE,0,&InvUpload.Expiration_Date},
      {CTRL_ENTRY,"Move_From",-1,1,0,1,0,32,1,0,NULL,"Move From",
         PROP_TEXT|PROP_REQ,25,&Work_Vars.MoveFrom},
      {CTRL_ENTRY,"Move_To_Site",-1,1,0,1,0,64,1,0,NULL,"Move To Site",
         PROP_TEXT|PROP_REQ|PROP_DISABLE,25,&Work_Vars.MoveTo_Site},
      {CTRL_ENTRY,"Move_To",-1,1,0,1,0,128,1,0,NULL,"Move To",
         PROP_TEXT|PROP_REQ,25,&Work_Vars.MoveTo},
      {CTRL_ENTRY,"Quantity",-1,1,0,1,0,256,1,0,NULL,"Quantity To Move",
         PROP_INTEGER|PROP_REQ|PROP_REUSEDEFAULT,11,&InvUpload.Quantity,"1"},
      {CTRL_HOTKEY,"Std_Hotkey_Exit",-1,1,0,-1,1,-1,0,F10,NULL,"F10=Exit"},
      {CTRL_ALIAS,"User_ID_A01",-1,2,0,0,0,1,1,S_User_ID,NULL,"le User ID"},
      {CTRL_ALIAS,"Site_A01",-1,2,0,0,0,2,1,S_Site,NULL,"le Site"},
      {CTRL_HOTKEY,"Hotkey01",-1,1,11,-1,1,128,0,F2,NULL,"F2=Site"},
      {CTRL_CALC_TEXT,"Show_ItemNum_and_Description",-1,-1,0,0,0,-8,2,0,Calc_Show_ItemNum_
and_Description,""},
      {0}                                                // Null struct to mark end of array
   };

   static int aiFieldsequence[] = {-1,0,1,2,3,4,5,6,7,8,-2};
   //---> Begin Hook:After local variables for Actions and screen
   //-->After Local Variable for Actions and Screen code hook insertion point
   //---> End Hook

   PROCESSWINDOW                                           // Entry Processing Loop
   {
      //---> Begin Hook:Top of PROCESSWINDOW loop
      //-->Top of PROCESSWINDOW Loop code hook insertion point
      //---> End Hook
      if(upghotkey(S_Std_Hotkey_Exit))                     // Hotkey {F10}
      {
         break;
      }

      if(upghotkey(S_Hotkey01))                            // Hotkey {F2}
      {
         continue;
      }

      //---> Begin Hook:After Hotkey processing
      //-->After Hotkey Processing code hook insertion point
      //---> End Hook

      switch(aiFieldsequence[zWindowinfo.iThisfieldix])
      {
         case -1:                                          // INIT Event
            //---> Begin Hook:Top of INIT Event
            //-->Top of INIT Event code hook insertion point
            //---> End Hook
            readinisettings("Inv_Move_Inventory");        // Update properties from INI file
            STRUCTCLEAR(InvUpload);
            STRUCTCLEAR(Work_Vars);
            //---> Begin Hook:Bottom of INIT Event
            //-->Bottom of INIT Event code hook insertion point
            //---> End Hook
            break;

         case 0:                                           // User_ID
            if(fieldinput())
            {
            }
            break;

         case 1:                                           // Site
            if(fieldinput())
            {
               STRUCTCLEAR(Location);
               strcpy(Location.Site,InvUpload.Site);
               if(upgvalidatelookup(0,Location_File,Location_Key02) > 0)
            strcpy(InvUpload.Site,Location.Site);
         }
         break;

      case 2:                                         // Item_Number
         if(fieldinput())
         {
            STRUCTCLEAR(Item);
            strcpy(Item.Item_Number,InvUpload.Item_Number);
            if(upgvalidatelookup(VLOOK_DONTALLOWOVERRIDE,Item_File,Item_Key01) > 0)
               strcpy(InvUpload.Item_Number,Item.Item_Number);
         }
         break;

      case 3:                                         // Lot_Number
         if(fieldinput())
         {
         }
         break;

      case 4:                                         // Exp_Date
         if(fieldinput())
         {
         }
         break;

      case 5:                                         // Move_From
         if(fieldinput())
         {
         }
         break;

      case 6:                                         // Move_To_Site
         if(fieldinput())
         {
            STRUCTCLEAR(Location);
            strcpy(Location.Site,Work_Vars.MoveTo_Site);
            if(upgvalidatelookup(0,Location_File,Location_Key02) > 0)
               strcpy(Work_Vars.MoveTo_Site,Location.Site);
           fieldpropset(S_Move_To_Site,PROP_DISABLE,True);  // Disable After Entry Option
         }
         break;

      case 7:                                         // Move_To
         if(fieldinput())
         {
         }
         break;

      case 8:                                         // Quantity
         if(fieldinput())
         {
         }
         break;


     case -2:                                      // Processing After Last Input Field
         //---> Begin Hook:Top of PROCESS After Last Input Field
```

```
            //-->Top of PROCESS After Last Input Field code hook insertion point
            //---> End Hook
            sprintf(gsCalcresult,"MF");                    // Set_MoveFrom_TransType
            strcpy(InvUpload.Trans_Type,gsCalcresult);
            sprintf(gsCalcresult,"%s",Work_Vars.MoveFrom); // Set_MoveFrom_Location
            strcpy(InvUpload.Location,gsCalcresult);
            gdCalcresult = asciiadd(InvUpload_File);       // Add_MoveFrom_Record
            sprintf(gsCalcresult,"MT");                    // Set_MoveTo_TransType
            strcpy(InvUpload.Trans_Type,gsCalcresult);
            sprintf(gsCalcresult,"%s",Work_Vars.MoveTo);   // Set_MoveTo_Location
            strcpy(InvUpload.Location,gsCalcresult);
           sprintf(gsCalcresult,"%s",InvUpload.Site);    // Save_Upload_Site_In_Case_MoveTo_
Site_Used
            strcpy(Work_Vars.Save_Site,gsCalcresult);
            if(*Work_Vars.MoveTo_Site)                     // If01
            {
               sprintf(gsCalcresult,"%s",Work_Vars.MoveTo_Site);  // Set_MoveTo_Site
               strcpy(InvUpload.Site,gsCalcresult);
            }
            else                                           // Else_If01
            {
            }                                              // End_If01
            gdCalcresult = asciiadd(InvUpload_File);       // Add_MoveTo_Record
            sprintf(gsCalcresult,"%s",Work_Vars.Save_Site);// Restore_Original_Upload_Site
            strcpy(InvUpload.Site,gsCalcresult);
            fieldpropset(S_Move_To_Site,PROP_DISABLE,False);  // Enable_MoveFrom_Site_Field
            fieldgoto(S_Move_To_Site);
            *InvUpload.Item_Number = 0;
            *InvUpload.Lot_Number = 0;
            *InvUpload.Expiration_Date = 0;
            *Work_Vars.MoveFrom = 0;
            *Work_Vars.MoveTo_Site = 0;
            *Work_Vars.MoveTo = 0;
            InvUpload.Quantity = 0;
            fieldgoto(S_Item_Number);
            //---> Begin Hook:Bottom of PROCESS After Last Input Field
            //-- Bottom of PROCESS After Last Input Field code hook insertion point
            //---> End Hook
            break;
      }
   }
   ENDPROCESSWINDOW
}
```

B

APPENDIX - PROPERTY SHEETS

UPG Options

Use the UPG Options form to modify the way your UPG Work Environment operates.

Field/Control	Description
Reload on Startup	Enable this feature if you would like to have the ToolType you were working on during your previous session of UPG automatically loaded.
ToolBar	If this feature is enabled, a toolbar with many of the most commonly used UPG operations available will appear at the top of your UPG Work Space.
Balloon Help	Check this box to turn balloon help on where available.
Reset Messages	Press the Reset Messages button to activate all messages that have had the Don't Show This Message checkbox enabled. (This is useful if a user new to UPG will begin working with a *used* installation.)
Reset All	Press the Reset All button to restore UPG to the default installation settings. This will include color choices, compiler modifications/settings, and the recent file list.
Compiler	UPG comes pre-configured to use the integrated Power C compiler. If you prefer to use Borland C compiler v4.5 (or greater) or Microsoft Visual C++ v1.0 (or greater) as your compiler, select the appropriate compiler from this pull-down list.

EXE Compressor

Your UPG package includes WWPACK. This EXE compressor will automatically be used by UPG.

If you would like to use another EXE compressor, please enter the executable name of the compressor in this field. If your EXE compressor is not included in the Path statement of your AUTOEXEC.BAT, you will need to enter the fully qualified path of your executable (i.e. c:\compress\compress.exe).

Source Editor

UPG uses Windows Notepad as the default source editor. If you have an editor you would like to use with UPG, enter the executable in the Source Editor field. If your editor is not included in the Path statement of your AUTOEXEC.BAT, you will need to enter the fully qualified path of your executable (i.e. `c:\editor\editor.exe`).

Memory Model

The memory model determines that way your UPG generated application accesses your portable's memory. For example, a medium memory model for the integrated Mix Power C compiler will allow both the data and code areas of an application to be as large as your portable's available memory. This is an advanced operation and should only be modified by an experienced programmer. The default selection is the "best" selection for most circumstances.

Comment Style

Defines the character the denotes a comment in the UPG generated source code. If the user selects "`//`" from the pull down list, a comment in the source code will be preceded by "`//`." However, if the user selects "`/*`," a comment will have "`/*`" at the beginning and the end of the comment.

Stack (k)

Stack is the memory allocated to application variables. Unless you are very experienced in modifying stack values with other programming tools, do not modify the default value. The default value has been optimized for supported portable data collection units.

Heap (k)

Heap is the area of memory that is reserved for the application's use. Unless you are very experienced in modifying heap values with other programming tools, do not modify the default value. The default value has been optimized for supported portable data collection units.

Compiler Batch Command

Creates the batch file that runs the compiler, linker, or make utility that is used to create your executable. This batch file is named UPGBUILD.BAT and is stored in the UPG directory.

Compiler Project Commands

Creates the project file for your application. A caret (^) denotes a hard return (end of line).

Compiler Debug Command

Specify the executable of the debugger you are using. Make sure that this executable is in the Path statement of your Autoexec.bat file or enter a fully qualified path name in this field (i.e. - c:\debug\debug.exe).

Debug Info Strip Command

Enter the command for your debugger to remove the Debug Info placed in your EXE file. This Debug Info will be removed when your program your portable. It is especially important to remove Debug Info when you are working with large applications (greater than 150K).

File/Variable Properties

The File/Variable ToolType, as the name suggests, allows you to define the Files and Variables that are required by your portable application. Similar to other ToolTypes, basic properties of the File/Variable ToolType will be defined in the Properties window and more advanced features will be defined in the Options window.

Field/Control	Description
File Types	UPG version 1.0 supports the following four file types: **ASCII File** - Creates a user defined delimited field. Users have the option of defining their delimiter and specifying whether or not to enclose text inside of quotes. **Fixed Length File** - This selection creates a file with fixed length fields and does not require a delimiter. **Memory Variables Group** - This file type creates a global C structure that can store information in your portable's volatile memory (the data stored in this file will be lost when you exit your portable data collection application). **INI File (Persistent Variables)** - Stores information in an INI file between application sessions. Use this file to store information that you might store in a variable, but do not want to be lost when you exit the portable data collection application.

Name

After you have selected a file type, you need to define the name for this file. This name will label the tool that will appear when you expand the **File/Variable** Tool-Type. Similar to naming other tools, you must limit the name to 32 characters in length and only use valid C naming conventions (if an invalid character is used, the character will be converted to an underscore).

Note: If the name of your tool contains the word "Upload," the name of your file will default to UPLOAD.TXT; for example, if you name your tool, "InvUpload," the name of your file will be UPLOAD.TXT. If you need to specify another file name, you can change the name of the file to suit your needs.

File Name

When you name the tool, the file name will automatically be derived from the value specified in the Name field when you press the <ENTER> or <TAB> keys (it will consist of the first eight (8) characters of the tool name). If you want to define another file name, place your cursor in the File Name field and edit the file name to meet your needs.

Note: You do not need to enter the three letter extension; it will automatically be added for you. If you are building an ASCII or Fixed Length file, .txt will be added to your file name. If you are building a Persistent INI file, .ini will automatically be added to your file name.

Optional Description

You can enter a description of your file in the **Optional Description** field. The field is a rough equivalent of a comment in a line of source code.

Fields

Once a file has been defined, you use the **Fields** window to define the fields that will make up your file. When you create the fields, you will want to pay careful attention to the order that you create the fields; the order the fields appear in the Fields window is the order that fields are placed in the file. The order of your fields may become an issue if you will be transferring data from your portable to an application on your desktop (host) PC.

Field Properties

Your ASCII, Fixed Length, and Memory Variable files are made up of fields, or individual containers that hold unique pieces of data (The Random House Personal Computer Dictionary defines a field as "a space allocated for a particular item of information"). A series of related fields make a record.

Field/Control — **Description**

Field Name

The name of your field must comply with C naming conventions; the value that you place in the **Field Name** field will appear when you are defining Input Fields.

Field Type

Your field may be one of several types:
Text - This field type will accept ASCII text up to 128 characters in length (includes alpha, numeric, and punctuation characters).
Integer - An integer field can only accept whole numbers and has a maximum input size of 11 numbers. It is declared as a type `long` in the UPG generated source code.
Floating Point - Will accept decimal (real) numbers. It is declared as a type `double` in the UPG generated source code.
Date - This field will only accept a date value and its maximum size is automatically calculated (the maximum length cannot be altered).
Time - This field will only accept a time value and its maximum size is automatically calculated (the maximum length cannot be altered).
Date Stamp - This field automatically inserts the current date into the field value and its maximum size is automatically calculated (the maximum length cannot be altered).
Time Stamp - This field automatically inserts the current time into the field value and its maximum size is automatically calculated (the maximum length cannot be altered).
Yes/No - This field type accepts two values; a yes or a no. A yes value write a value of 1 to your data file, while a value of no writes a value of 0 to your data file. If you modify your `application.ini` file, you can define multiple yes/no characters; for example, you may want a value of yes returned if the user presses a <Y> or <1> on the keyboard (<Y> is the default character for yes and <N> for no).

Max Length

Enter the maximum length of entry that you want this field to accept. (You can define the maximum length for Text, Integer, and Floating Point fields only; the other field types automatically set the maximum length).

Optional Description

You may place descriptive text in the **Optional Description** field for documentation purposes. The description field is essentially a comment field for users to use at their own discretion.

Program Properties

The Program ToolType allows you to define the name of your application’s executable (.EXE) file, and define the associated parameters, such as Program Description, Startup Tool, Force Configuration Download, and, for the advanced users, Code Hooks.

The basic properties of your program are set in the Program Properties window. The following details will help you better define your portable application to fit your individual requirements.

Field/Control	Description
Program Name	The name of your executable file needs to be six (6) characters or less (UPG adds two characters to your filename to manage the different files that are required to create your portable application). If you type more than six (6) characters, you will receive an error message telling you that you have entered an invalid filename. You do not need to add the .exe extension to your filename; UPG will automatically add it for you.
Optional Description	If you would like to add a description for documentation purposes, place this description in the description field. The description field is essentially a comment field for users to use at their own discretion.
Startup Tool	Define the tool that is executed when you launch your portable application via the Startup Tool field (in most cases, this will be a menu). You can select an existing tool from the pull down list, accept the default tool (this will be your program name_main), or define your own. If you accept the default or define your own tool, you will see the tool appear in both your ToolBox and Program pane with the phrase (To Do) attached to the end of your tool. (To Do) denotes that you have an outstanding feature to complete.

Form: Menu Template Properties

Use the Menu with (or without) Exit Key templates to create a blank menu screen that can contain text, aliases, hotkeys, or display fields. Depending on the template selected, an exit key (F10=Exit) may automatically be placed at the bottom of your screen.

Field/Control	Description
Name	After you have selected a form template, you need to define the name for this form. This name will label the tool that will appear when you expand the **Form** Tool-Type. Similar to naming other tools, you must limit the name to 32 characters in length and only use valid C naming conventions (if an invalid character is used, the character will be converted to an underscore). If you double click on a (To Do) item in the Program pane and select the Form menu item, the name of your (To Do) tool will automatically be placed in the Name field. This automation will ensure that the form you are creating will automatically be connected to the calling tool, such as another form.
Optional Description	If you wish to assign a description to this form, place the descriptive text in the Optional Description field. The value placed in this field works in a similar manner as a comment entry in a line of code.

Screen

Position your cursor within the Screen window, right-click and select Add or press <CTRL><A>, and select the control you would like to add to your form. The controls listed below are available when working with a Menu template.

- ❑ **Text** - The Text control will place a text label on your form.
- ❑ **Hotkey** - The Hotkey control will define a hotkey to activate a specific tool. If you specify a tool that does not yet exist, this new tool will appear in your Program pane with the (To Do) annotation.
- ❑ **Display Field** - This screen control displays a value. This value, for example, can be the result of a UPG Action or the value contained in a lookup field.
- ❑ **Alias** - The Alias control creates a representation of an existing control (i.e. Input Field) that operates the same as the original control; this is useful when creating an application for multiple portable types and eliminates the need for creating duplicate controls.

Form: Message Box Template Properties

Use the Message Box with (or without) Anykey Message templates to create a message box that can contain only text, aliases, or display fields and are dismissed by pressing any key. Depending on the template selected, the message "Any Key To Continue" may appear at the bottom of this screen.

Use the Message Box View File template to build forms to display collected data. The tool you create with this form should be used with the View/Edit File utility (select this tool from the Edit Tool pull-down list when defining your View/Edit File utility).

Field/Control	Description
Name	After you have selected a form template, you need to define the name for this form. This name will label the tool that will appear when you expand the **Form** Tool-Type. Similar to naming other tools, you must limit the name to 32 characters in length and only use valid C naming conventions (if an invalid character is used, the character will be converted to an underscore). If you double click on a (To Do) item in the Program pane and select the Form menu item, the name of your (To Do) tool will automatically be placed in the Name field. This automation will ensure that the form you are creating will automatically be connected to the calling tool, such as another form.
Optional Description	If you wish to assign a description to this form, place the descriptive text in the Optional Description field. The value placed in this field works in a similar manner as a comment entry in a line of code.

Screen

Position your cursor within the Screen window, right-click and select Add or press <CTRL><A>, and select the control you would like to add to your form. The controls listed below are available when working with a Message Box template.

- ❑ **Text** - The Text control will place a text label on your form.
- ❑ **Display Field** - This screen control displays a value. This value, for example, can be the result of a UPG Action or the value contained in a lookup field.
- ❑ **Alias** - The Alias control creates a representation of an existing control (i.e. Input Field) that operates the same as the original control. This is useful when creating an application for multiple portable types and eliminates the need for creating duplicate controls.

Form: Multipage Sequence Template Properties

Allows you to define a data collection sequence; you can define multiple pages (separate screens) to collect specific pieces of data. "F10=Exit" is automatically placed at the bottom of your multipage sequence to allow your users to exit the data collection sequence.

Field/Control	Description
Name	After you have selected a form template, you need to define the name for this form. This name will label the tool that will appear when you expand the **Form** Tool-Type. Similar to naming other tools, you must limit the name to 32 characters in length and only use valid C naming conventions (if an invalid character is used, the character will be converted to an underscore). If you double click on a (To Do) item in the Program pane and select the Form menu item, the name of your (To Do) tool will automatically be placed in the Name field. This automation will ensure that the form you are creating will automatically be connected to the calling tool, such as another form.
Optional Description	If you wish to assign a description to this form, place the descriptive text in the Optional Description field. The value placed in this field works in a similar manner as a comment entry in a line of code.

Screen

Position your cursor within the Screen window, right-click and select Add or press <CTRL><A>, and select the control you would like to add to your form. The controls listed below are available when using a Multipage Sequence template.

❑ **Text** - The Text control will place a text label on your form.

❑ **Hotkey** - The Hotkey control will define a hotkey to activate a specific tool. If you specify a tool that does not yet exist, this new tool will appear in your Program pane with the (To Do) annotation.

❑ **Input Field** - The Input Field control allows the user to enter data into the portable application. When the user defines an Input Field control, the field is attached to a field that is defined using the File/Variable ToolType.

❑ **Display Field** - This screen control displays a value. This value, for example, can be the result of a UPG Action or the value contained in a lookup field.

❑ **Alias** - The Alias control creates a representation of an existing control (i.e. Input Field) that operates the same as the *aliased* control; this is useful when creating an application for multiple portable types and eliminates the need for creating duplicate controls.

Utility Properties

When you have finished collecting data with your portable data collection application, there may be a number of different operations you want your application to perform, such as sharing the collected data between your portable and a PC. The UPG Utility ToolType provides the miscellaneous operations that you will require to complete the feature set of your portable data collection application.

As of UPG v1.0, the following utilities are available:

Field/Control	Description
Send File to Host	This utility allows you to transfer a data file from your portable data collection unit to a target PC.
Receive File from Host	Use this utility when you need to transfer a data file from a host PC to your portable data collection unit.
View/Edit File	The View/Edit File utility enables you to view and/or edit the contents of a file from within your portable data collection application.
Run External DOS Program	Allows the user to define an executable command to launch another DOS program on the portable data collection unit.
Exit to DOS	This utility closes the UPG generated application and returns the user to a DOS prompt.
Delete File	Allows the user to delete a specified file (useful when the user want to clear or rest the data collection file after sending the data to a target application).

Key

Defining a Key field will noticeably improve the performance of your application when searching a file for a particular record. This feature will most often be used when using validation files.

Field/Control	Description
Name	The name of your Key field must comply with C naming conventions.
Optional Description	If you wish to assign a description to this Key field, place the descriptive text in the Optional Description field. This description is for documentation purposes only.
Case Sensitive	Activating this feature will require the lookup value to match the case of the search value.
Key Fields	This window lists all of the fields that you have specified as Key fields (you will notice that when you specify a Key field, that field will disappear from the File Fields list).
File Fields	Contains all of the fields available in the file. When a file is specified as a Key field, it will be removed from this list (you can specify the same field as a key field twice).

Enable/Disable Properties

The Enable/Disable UPG Action component allows the user to turn on and off specific input fields. This feature also allows the user to move to a specific input field that is not in the order of the data collection sequence. (This is useful when a user wants to reset a value that is not prompted for during each pass of a multipage sequence, such as a Site field that has the Don’t Ask on Reuse option enabled.)

Field/Control	Description
Name	The name of your Enable/Disable Action component must comply with C naming conventions.
When to Execute	Select the event that will trigger your Loop Action Component. **Setup:** The Loop Component will be processed before your form is written to your screen. **Process:** Occurs at the end of a data collection sequence, before you return to the loop field (specified in the Properties tab of your Multipage Sequence). **Exit:** Occurs when you press the exit key to leave the current screen. **After Input Field:** (If this is selected, another field will appear; select the input field that you want the Loop Action Component to run after). Happens when user presses <ENTER> or the input device sends the return character. **Called by Other Control:** Is run when another control calls the Loop Action Component.
Input Field to Execute After	If **After Input Field** was selected from the When to Execute pull-down list, select the input field to specify which input field this action component is to execute after.
Input Field to Enable/Disable	Select the Input Field that will be the target of the Enable/Disable Action component.

Action

Select the event that you want to occur when the action component is executed.
Disable: The target input field will not be able to accept data entry.
Enable: Target input field is able to accept data entry.
Toggle: Reverses the current Enable/Disable state of the input field. For example, if an input field is currently disabled and Toggle is selected, the input field will be enabled when the Enable/Disable action component is executed.

Goto Field After Change

Makes the target input field the active input field. This is a convenient method to go to an input field that is outside the current multipage sequence order (allows the user to jump around in the collection sequence).

Make New State Persistent

Enable this checkbox to make the new Enable/Disable state persistent. The next time the user opens the multipage sequence that contains the target input field, the Enable/Disable state will be retained.

Text Properties

The Text control will allow you to place static text on your display; This control will most commonly be used for titles and text messages.

Field/Control	Description
Name	The name of your Text control must comply with C naming conventions.
Display Text	Enter text that you want to be displayed. Remember to keep your text length less than your PDT display width (watch your **Length** counter above the **Display Text** field).

Input Field Properties

Input fields accept data entry from the users of your portable data collection application.

Field/Control	Description
Name	The name of your Input Field must comply with C naming conventions.
Input Field	Select the field that you want to enter data into. If you have not defined a File/Variable tool, you will not be able to select a field (you must define your storage structure before you begin building the interface by which you will enter data, otherwise, you wont have any targets for your data).
Display Text	After you select your Input Field, the Display Text will automatically be derived from your Input Field. If this derived text does not meet your program requirements, you can change the text. For example, if you select the Input Field "Item.ItemNumber," then the derived display text will be "Item Number."
Required	If this feature is activated, it will require the user to enter a value in this field before proceeding to the next input field defined in the sequence.
Reuse Value	If this field is in a looping data collection sequence and Reuse Value is enabled, the previously entered value in this field will automatically be used during subsequent passes through the loop.

Hotkey Properties

A hotkey allows you to assign a tool (a form or utility) to an <F> key on your PDT.

Field/Control	Description
Name	Enter the name of your Hotkey in the Name field. Remember, this control name must comply with C naming conventions.
Hotkey	Select the hotkey from the **Hotkey** pull-down list you want to assign the called tool to.
Display Text	Enter text that describes what tool the <F> key calls. Remember to keep your text length less than your PDT display width (watch the **Length** counter above the **Display Text** field).
Tool	UPG will automatically create a tool name derived from your Display Text entry; if this tool exists, the <F> key will run the specified tool. If the tool does not exist and you accept the tool name, a (To Do) will be created in your Program window (double-click on the To Do to build a tool). If you want to run an existing tool with the <F> key, select the existing tool from the **Tool** pull-down list.

While Condition is True Loop Properties

The Loop Action component repeats a user defined action component until a certain condition is met.

Field/Control	Description
Name	Enter the name of your Loop Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
Calculation Components	Contains the specified condition for this loop. Use the Expression Builder to construct the condition statement that must be met.
When to Execute	Select the event that will trigger your Loop Action Component. **Setup**: The Loop Component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field (specified in the Properties tab of your Multipage Sequence). **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: (If this is selected, another field will appear; select the input field that you want the Loop Action Component to run after). Happens when user presses <ENTER> or the input device sends the return character. **Called by Other Control**: Is run when another control calls the Loop Action Component.
UPG Syntax	Loop End_Loop

C Syntax	`while(`*`condition`*`)`
	`{`
	`}`

Usage

Use the While Condition is True loop to execute a UPG Action component (or a series of UPG Action components) while the condition returns a value of true (non-0). When the condition returns a value of false (0), the specified UPG Action components will not be executed.

Until Condition is True

The Until Condition is True loop creates a WHILE NOT structure:

Field/Control	Description
Name	Enter the name of your Loop Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
Calculation Component	Contains the specified condition for this loop. Use the Expression Builder to construct the condition statement that must be met.
When to Execute	Select the event that will trigger your Loop Action Component. **Setup**: The Loop Component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field (specified in the Properties tab of your Multipage Sequence). **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: (If this is selected, another field will appear; select the input field that you want the Loop Action Component to run after). Happens when user presses <ENTER> or the input device sends the return character. **Called by Other Control**: Is run when another control calls the Loop Action Component.
UPG Syntax	Loop End_Loop

C Syntax

```
while(!condition)
{
}
break;
```

Note: The exclamation mark (!) is automatically placed in the condition statement of your UPG generated source code. Do not insert the exclamation mark (!) in your condition statement.

Usage

Use the Until Condition is True loop to execute a UPG Action component (or a series of UPG Action components) while the condition returns a value of false (0). When the condition returns a value of true (non-0), the specified UPG Action components will not be executed.

For Range of Values

The For Range of Values loop creates a FOR structure:

Field/Control	Description
Name	Enter the name of your Loop Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Loop Action Component. **Setup**: The Loop Component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field (specified in the Properties tab of your Multipage Sequence). **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: (If this is selected, another field will appear; select the input field that you want the Loop Action Component to run after). Happens when user presses <ENTER> or the input device sends the return character. **Called by Other Control**: Is run when another control calls the Loop Action Component.
Loop Variable	The variable being incremented.
From	The initialization value of the Loop Variable
To	Specifies the maximum number; when the maximum number is exceeded, the loop will terminate.
Step	Specifies the increment value (defaults to 1).
UPG Syntax	Loop End_Loop

C Syntax	`for(`*`initialize`*`, `*`condition`*`, `*`increment`*`)`
	`{`
	`}`

Usage

Use the For Range of Values loop to execute a UPG Action component (or series of UPG Action components) a specified amount of times.

Loop Variable - Specifies the field that is initialized, that the condition is checked against, and is incremented.

From - Specifies the *initialization* value (i.e. Application.Qty = 1, where "1" is the From value).

To - Specifies the value of the condition (i.e. Application.Qty <= 10, where "10" is the To value).

Step - Specifies what the increment value is (i.e. Application.Qty += 2, where "2" is the Step value).

Beep Properties

Use the Beep process component to produce an audible tone that is called by a process.

Field/Control	Description
Name	Enter the name of your Beep process component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Beep	Select the event that will trigger your Beep Action Component. **Setup**: The Beep Component will be processed before your display is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field (specified in the Properties tab of your Multipage Sequence). **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: (If this is selected, another field will appear; select the input field that you want the Beep Action Component to run after). Happens when user presses <ENTER> or the input device sends the return character. **Called by Other Control**: Is run when another control calls the Beep Action Component.

If (IF-THEN-ELSE) Properties

Similar to conventional programming, you can use the IF Action Component to build an IF-THEN-ELSE condition.

If the field or variable meets the condition (TRUE), then the statement that comes directly after the IF statement (a) will be performed. If the file or variable does not meet the condition (FALSE), then the statement after the THEN statement (b) will be performed. For more details, you should review IF-THEN-ELSE condition statements in a C programming reference book.

Field/Control	Description
Name	Enter the name of your IF Action Component in the **Name** field. Remember, this control name must comply with C naming conventions. (Tip: You may want to leave "IF" at the beginning of your name so that you can easily identify the IF statement in your Action window.)
Condition	Contains the specified condition for this IF-THEN-ELSE structure. Use the Expression Builder to construct the condition statement that must be met.
When to Execute	Select the event that will trigger your IF Action Component. **Setup**: The IF Action Component will be processed before your display is written to your screen. For example, when you… **Process**: Occurs at the end of a data collection sequence, before you return to the loop field (specified in the Properties tab of your Multipage Sequence). **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: (If this is selected, another field will appear; select the input field that you want the IF Action Component to run after). Happens when user presses <ENTER> or the input device sends the return character. **Called by Other Control**: Is run when another control calls the IF Action Component.

Program Options

The Program Options tab contains the following optional settings:

Field/Control	Description
Force Configuration Download	If this feature is activated, then your portable application will require a valid UPG.INI file to be loaded on your portable unit (the UPG.INI file is checked when the portable is programmed and when the application is started). If this feature is activated and no valid UPG.INI file exists on the portable, then the user will be required to load a valid UPG.INI file before the portable application will operate correctly.
Program Restart Timeout	Enter the time (in seconds) that will force your portable application to restart if no keystrokes have occurred on the portable data collection unit (this feature will most often be used when implementing security).

Program Advanced

The Program Advanced tab allows you to insert code hooks (embedded C code) into the UPG generated source code. Startup Tool Parameters can also be specified to pass to a calling tool.

Field/Control — **Description**

Startup Tool Parameters

This field allows you to send values to arguments defined in the Advanced tab of the Form definition screen; consequently, you can only place values in this field when you are calling a Form that has the appropriate arguments declared in the parameter list. (If this explanation is not sufficient, review Chapter 9 in the UPG User's manual or consult an experienced C programmer for more information on function parameters.)

Code Hooks

Lists the available insertion points into the current program tool's generated source code:

Note: Code hooks inserted directly into the program tool (`application.exe`) will be inserted in the `application.c` source code file (this file will not have a numerical suffix), such as `inv.c`.

Start of Program File: Inserts your code immediately before the Main function. You can only insert directives and global declarations in this code hook.

After { in Function Main: Inserts your code immediately after the opening brace ({) in the Main function. You can insert both local declarations and statements (declarations must be defined first).

Before } in Function Main: Inserts your code immediately before the closing brace (}) in the Main function. You can only place statements in this code hook insertion point.

If your C code in a Code Hook references a UPG tool that is not explicitly linked via a UPG generated control, you will need to add the tool to your application by highlighting the tool and selecting **Add Unattached Tool** from the **Program** menu (accessed via the **Advanced** sub-menu).

File/Variable Options

The File/Variable ToolType, as the name suggests, allows you to define the Files and Variables that are required by your portable application. Similar to other ToolTypes, basic properties of the File/Variable ToolType will be defined in the Properties window and more advanced features will be defined in the Options window.

Field/Control	Description
Create	If this feature is activated, the file currently being defined will be created if it does not exist when your portable application tries to access the file.
Quotes	Turning Quotes on will enclose textual data (string fields) within quotes; for example, your data may look like 3, "Text Description", 42.4
Delimiter	You can specify any single character as your field delimiter. Most often, a field delimiter is a comma (,), but UPG will handle ASCII character as a field delimiter. Be careful, however, when you are defining your delimiter. Since UPG is a Windows product, it will accept any characters from the Windows ANSI character set, and these characters may not translate correctly to a DOS ASCII character set.

The following parameters are available to modify the way input fields operate.

Field/Control	Description
Default Value	This value will automatically be placed in the Input Field.
Reuse Default Each Time	Each time you come to this input field, it will reuse the default value, regardless if you have previously entered a value during the current collection sequence.

Don’t Ask on Reuse If you enable this control, the Input Field will automatically be populated with the default value on the first pass through the Multipage Sequence. The user may overwrite the default value if desired. During your second pass, however, this field will automatically be populated with the default value and the user will be automatically advanced to the next field. (After the first pass, the user will not be able to edit the value in this Input Field).

Data Identifier A string of characters that precede a bar code that identifies the type of data. For example, a bar code may have the value "PN123-76." The prefix "PN" identifies that this bar code is a part number. Place the desired data identifier in the **Data Identifier** field.

Data ID Required Enable this checkbox to require data entered in this field to contain the specified Data Identifier (see above). If the identifier does not exist on the entered data, the value will be rejected.

Strip Data ID from Input If this checkbox is enabled, the Data Identifier will be removed from the entered data before it is written to your data file. For example, if "PN" was your data identifier and you entered the value "PN123-76," only "123-76" will be written to your data file.

Field Position This control determines the placement of your input field on your Multipage Sequence screen:
Same Line - Position the Input Field to the right of your Display text.
Next Line - Positions the Input Field under your Display Text.

Disabled by Default Enabling this checkbox makes this Input Field disabled by default. In order for this field to be enabled, the user will need to enable it by using the Enable/Disable action component (this will enable the field when a determined condition is met).

Disabled after Entry After you enter a value into this field during the first pass through the collection sequence, the Input Field will be disabled for all subsequent passes. Use the Enable/Disable Action Component to re-enable this field (field will be enabled when a determined condition is met).

Case **As Entered** - Data will be displayed and saved to disk using the case that is entered.
Upper Case - Data will be displayed and saved to disk using only upper case characters, regardless of the case when entered.
Lower Case - Data will be displayed and saved to disk using only lower case characters, regardless of the case when entered.

Password Asterisks (*) will be echoed to the screen during data entry. This feature, of course, will most often be used when collecting a user password.

Screens

The basic principle to remember when building displays for multiple portables and languages is that your currently selected control will appear on selections that are marked with a caret (➤). For example, if the Percon Falcon screen, pages 2 and 3, and English selections are marked with a caret, than the current control will appear on pages 2 and 3 of your multipage sequence when your application is compiled in English for the Percon Falcon.

Field/Control	Description
Screens	Lists the supported portable displays. Place a caret next to the display that you want the currently selected control to appear on (all screens are selected by default).
Row	Specifies the "row" position of your currently selected control. This value refers to the top-left edge of your control (defaults to the next available row, starting with row 1).
Column	Specifies the "column" position of your currently selected control. This value refers to the top-left edge of your control (defaults to column 1).
Position	There are two types of selection available in the Position pulldown list (defaults to As Placed). **As Placed** - This selection will position your control based on the values in the Row and Column fields. **Bottom** - This selection, along with its variations, will dynamically place your control on the bottom of your screen (or the specified rows above the bottom of your screen), regardless of your display size.
Pages	The values in this window coincide with the page numbers of your multipage sequence. Place a caret next to all of the pages that you want the currently selected control to appear on. For example, if you want a display field to appear on pages 2, 3, and 4 of your multipage sequence, place a caret next to the 2, 3, and 4 list items (defaults to the page where control is created).

Languages Place a caret next to the languages that you want your currently selected control to appear in. This will determine what is displayed when you select a language from the Portable menu (defaults to English only).

Invert Reverses the current selection status. All items that are currently selected, will be deselected and all items that are currently deselected, will be selected.

None Deselects all list items in the window, regardless if the item is currently selected or not.

All Select all list items in the window, regardless if the item is currently selected or not.

Input Field Verify

Use the features in the Input Field Verify tab to control data input into your UPG generated portable data collection application; such features can include data verification, key-in patterns (for information like phone numbers, social security numbers, etc.), and specify data source.

Field/Control	Description
Lookup Field	Select the field from this pull-down list that will provide validation for the value entered in the current input field.

Note: UPG enables you to specify a hierarchical lookup, up to three levels deep. If a hierarchical lookup is defined, all values in the lookup targets must be validated against the respective lookup field (the currently selected input field is the Lookup Target 1) for the current value to be "valid."

Field/Control	Description
Lookup Field 2	Select a field from this pull-down list that you want to perform the secondary lookup. This lookup validates the entry entered in the field specified in the Lookup Source 2 field.
Lookup Field 3	Select a field from this pull-down list that you want to perform the third level lookup. This lookup validates the entry entered in the field specified in the Lookup Source 3 field.
Lookup Source 2	If a Lookup Field 2 is defined, the file selected from this pull-down list will be validated by the field selected in the Lookup Field 2 pull-down list.
Lookup Source 3	If a Lookup Field 3 is defined, the file selected from this pull-down list will be validated by the field selected in the Lookup Field 3 pull-down list.

Ignore if File Missing

Enable this checkbox if you want your UPG application to ignore any Lookup Field validation if the specified validation file is missing (this will enable your application to operate correctly if the validation file is missing); if this checkbox is disabled, then your application will require a proper validation file to operate correctly.

Allow Override

Enable this checkbox to allow a user to enter a value that is not contained in the validation file. If this checkbox is disabled, the user will only be able to enter values that are contained in the specified validation file.

Input Source

Specify the source of entry from the pull-down list. The following is a list of choices:
Any - Allows input from either the scanner or keyboard.
Scanner - Allows input from only the scanner (from the attached scanning device).
Keyboard - Allows input from only the portable data collection unit's keyboard.

Entry Pattern

Use the entry pattern feature to control the data entry pattern. For example, if you want to have a Social Security Number entry pattern, the user would specify ###-##-####. Specify your entry pattern using a combination of the Entry Pattern Building Blocks (see Chapter 5, Data Verification in your User's Manual for more information).

Valid Characters

List the characters in this field that the currently selected input field will accept as valid characters. Any character entered that is not in this list will immediately be rejected. If you want your alpha characters to be case sensitive, place a $ at the beginning of the list. If there are no values in this field, than all characters will be accepted (if an entry pattern is defined, entered values will still be subject to meet pattern requirements).

Hotkey Options

A hotkey allows you to assign a tool (a form or utility) to an <F> key on your PDT.

Field/Control	Description
Call Tool Parameters	This field allows you to send values to arguments defined in the Advanced tab of the Form definition screen. Consequently, you can only place values in this field when you are calling a Form that has the appropriate arguments declared in the parameter list. (If this explanation is not sufficient, review Chapter 9 in the UPG User's manual or consult an experienced C programmer for more information on function parameters.)
Exit after Execution	Enable this checkbox to exit the current form when this hotkey is activated.

Call a Tool Properties

The Call a Tool UPG Action component allows you to execute a tool based on the selection in the When to Process pull-down list, or it can be called from within a series of UPG actions, such as within an IF-THEN-ELSE condition statement.

Field/Control	Description
Name	Enter the name of your File Action Component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Process	Select the event that will trigger your File Action Component. **Setup**: The File Component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
Input Field to Execute After	If you select After Input Field from the When to Process pull-down list, this field will appear. Select the Input Field that you would like this component to execute (run) after.
Tool to Call	Specify the tool that you would like your Call a Tool Action component to execute.

Call a Tool Options

If you define a Call a Tool Action Component, you can specify values to pass to the tool being called. This requires that the tool being called to have the appropriate arguments declared in the parameter list.

Field/Control	Description
Call Tool Parameters	This field allows you to send values to arguments defined in the Advanced tab of the Form definition screen. Consequently, you can only place values in this field when you are calling a Form that has the appropriate arguments declared in the parameter list. (If this explanation is not sufficient, review Chapter 9 in the UPG User's manual or consult an experienced C programmer for more information on function parameters.)

Form Options

You will most likely use the Form ToolType more than any other UPG ToolType. This particular ToolType enables you to build the user interface for your portable application.

Field/Control	Description
Clear Screen on Entry	In certain circumstances, will blank out the screen when a page is reached (works just like the CLS DOS command).
Beep on Entry	If activated, this option will cause your portable unit to produce an audible tone when the form is first opened.
Clear File Buffer on Setup	Clears out all data in the file buffers and resets all values to null.
Use INI File for Advanced Options	If this feature is enabled, an `application.ini` file will be created for your portable application. For example, if you have a program that is named `Collect.exe` and this feature is enabled, you will also generate a `Collect.ini` file the first time you activate your multipage sequence. The user can edit this file to modify sequence order and maximum field lengths.
Add Record After Last Field	After the last field in a sequence has accepted a value, the record will then be added to the target file.
Change Record when Requested	Enable this checkbox if you want to use this multipage sequence to edit a file using the View/Edit File utility (select this multipage sequence from the Edit Tool pull-down list). If you use this form to edit data and this checkbox is disabled, modifications to existing data will not be saved.

UPG Actions

Using UPG Actions will provide you a platform to create advanced features for your UPG generated application. The window in the Actions tab contains a listing of defined Action components. Below you will find a table of UPG Action Components.

You can see a listing of available UPG Action components by positioning your mouse in the Actions or Calculations window and right-click; a menu of available action components will appear.

Action Components	Description
Calculate Text Values	Calculates or "builds" a text value or string that can be displayed on screen or saved to a memory variable or file.
Calculate Numeric Values	Calculates a numeric value that can be displayed on screen or saved to a memory variable or file.
Enable/Disable Input Field	Allows you to enable or disable a specific Input Field; this Action Component will most often be used in conjunction with the IF (IF-THEN-ELSE) Action Component. When combined with the IF component, you can enable/disable Input Fields based on a certain conditions.
IF (IF-THEN-ELSE)	Builds an IF-THEN-ELSE condition structure. If a condition is true, than a defined procedure will occur, whereas if the condition is false, another defined procedure will occur.
Loop	Enables you to repeat an event (or series of events) until a particular condition is true. There are three Loop types: While Condition is True, Until Condition is True, and For Range of Values (see online help for more information).
File	Performs several File related operations, such as file navigation, record counting, and deleting specific records.

Beep

Causes portable data collection unit to produce an audible tone ("beep") is a condition is true.

Call a Tool

Executes another tool based on the When to Execute time. This Action component, however, is most powerful when used with an IF or Loop Action component (you can execute different tools based on the result of the IF or Loop condition).

Form Advanced - Multipage Sequence

You will most likely use the Form ToolType more than any other UPG ToolType. This particular ToolType enables you to build the user interface for your portable application.

More advanced features are available in the Program Options window. Simply click on the Options tab to access the advanced features. (Please see Chapter 9 for more information.)

Field/Control	Description
Return Type	If necessary, specify the type of data that is returned from the current form.
Parameters	Define any arguments that you want to be placed in the current form's parameter list. Use the Call Tool Parameters field in the Options tab of a Hotkey or a Call a Tool Action component definition screen to pass values to this form's defined arguments. (If this explanation is not sufficient, please consult an experienced C programmer for more information on function parameters.)
Code Hooks (Multipage Sequence Forms)	Lists the available insertion points into the current form's generated source code (see Chapter 9 for more information).

If your C code in a Code Hook references a UPG tool that is not explicitly linked via a UPG generated control, you will need to add the tool to your application by highlighting the tool and selecting **Add Unattached Tool** from the **Program** menu (accessed via the **Advanced** sub-menu).

Form Advanced - Menu

You will most likely use the Form ToolType more than any other UPG ToolType. This particular ToolType enables you to build the user interface for your portable application.

More advanced features are available in the Program Options window. Simply click on the Options tab to access the advanced features. (Please see Chapter 9 for more information.)

Field/Control	Description
Return Type	If necessary, specify the type of data that is returned from the current form.
Parameters	Define any arguments that you want to be placed in the current form's parameter list. Use the **Call Function Parameters** field in the **Options** tab of a Hotkey definition screen to pass values to this function's defined arguments. (If this explanation is not sufficient, please consult an experienced C programmer for more information on function parameters.)
Code Hooks (Menu Forms)	Lists the available insertion points into the current form's generated source code (see Chapter 9 for more information).

If your C code in a Code Hook references a UPG function that is not explicitly linked via a UPG generated control, you will need to add the function to your application by highlighting the function and selecting **Add Unattached Function** from the **Program** menu (accessed via the **Advanced** sub-menu).

Form Advanced - Message Box

You will most likely use the Form ToolType more than any other UPG ToolType. This particular ToolType enables you to build the user interface for your portable application.

More advanced features are available in the Program Options window. Simply click on the Options tab to access the advanced features.

Field/Control	Description
Return Type	If necessary, specify the type of data that is returned from the current form's function.
Parameters	Define any arguments that you want to be placed in the current form's parameter list. Use the **Call Function Parameters** field in the **Options** tab of a Hotkey definition screen to pass values to this function's defined arguments. (If this explanation is not sufficient, please consult an experienced C programmer for more information on function parameters.)
Code Hooks (Message Forms)	Lists the available insertion points into the current form's generated source code (see Chapter 9 for more information).

If your C code in a Code Hook references a UPG function that is not explicitly linked via a UPG generated control, you will need to add the function to your application by highlighting the function and selecting **Add Unattached Function** from the **Program** menu (accessed via the **Advanced** sub-menu).

Calculate Text Value Component

Use the Calculate Text Value Action component to build expressions that return a text value.

Field/Control	Description
Name	Enter the name of your Calculate Text Value Action Component in the **Name** field. Remember, this control name must comply with C naming conventions.
Show Result on Screen	Will display the result of the Calculate Text Value on your portable display. The result will be displayed according to the settings in the Screens tab.
When to Execute	Select the event that will trigger your Calculate Text Value Action Component. **Setup**: The Calculate Text Value Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
Display Text	If you enabled the Show Result on Screen checkbox, this field will also become enabled; enter the display text that you would like to appear with the result of your Calculate Text Value Action component. Use the Field Position pull-down list in the Options tab to define the position of your display text.

Input Field to Execute After

If you select After Input Field from the When to Process pull-down list, this field will appear. Select the Input Field that you would like this component to execute (run) after.

Result Field

The result of your Calculate Text Value Action component will be saved in this field.

Calculation Components

Used to build/display calculation. Right-click in window to add an expression or operator. Use the Expression Builder when adding expressions to construct the calculation.

Calculate Numeric Value Component

Use the Calculate Numeric Value Action component to build expressions that return a numeric value.

Field/Control — **Description**

Name

Enter the name of your Calculate Numeric Value Action Component in the **Name** field. Remember, this control name must comply with C naming conventions.

Show Result on Screen

Will display the result of the Calculate Numeric Value on your portable display. The result will be displayed according to the settings in the Screens tab.

When to Execute

Select the event that will trigger your Calculate Numeric Value Action Component.
Setup: The Calculate Numeric Value Action component will be processed before your form is written to your screen.
Process: Occurs at the end of a data collection sequence, before you return to the loop field.
Exit: Occurs when you press the exit key to leave the current screen.
After Input Field: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below).
Called by Other Control: Is run when another control calls the File Action Component.

Display Text

If you enabled the Show Result on Screen checkbox, this field will also become enabled. Enter the display text that you would like to appear with the result of your Calculate Numeric Value Action component. Use the Field Position pull-down list in the Options tab to define the position of your display text.

Input Field to Execute After

If you select After Input Field from the When to Process pull-down list, this field will appear. Select the Input Field that you would like this component to execute (run) after.

Result Field

The result of your Calculate Numeric Value Action component will be saved in this field.

Calculation Components

Used to build/display calculation. Right-click in window to add an expression or operator. Use the Expression Builder when adding expressions to construct the calculation.

View/Edit File Utility

Use the Calculate Numeric Value Action component to build expressions that return a numeric value.

Field/Control	Description
Name	Enter the name of your View/Edit File utility in the Name field. Remember, this control name must comply with C naming conventions.
Optional Description	If you would like to add a description for documentation purposes, place this description in the description field. The description field is essentially a comment field for users to use at their own discretion.
File	Select the file that will be viewed and or edited by the tool created with this Utility template.
Edit Function	Select a multipage sequence that you would like to use to edit the data contained in the file selected from the File pull-down list. You will most often select the multipage sequence used to collect the data.
Display Function	Select a tool created with the Message Box View File Form template to view the data contained in the file specified in the pull-down list File pull-down list.

Action: Add Record to File (File)

Use the Add Record to File Action component to write the information currently stored in the file buffer to the target file. This Action component will most often be used when you have disabled the Add Record After Last Field checkbox in the Options tab of a Form definition screen.

Field/Control	Description
Name	Enter the name of your Add Record to File Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Add Record Action component. **Setup**: The Add Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.

Action: Lookup Record (File)

Use the Lookup Record Action component to lookup a record. This Action component works in a similar manner as field validation in the Verify tab of an Input Field control.

Field/Control	Description
Name	Enter the name of your Lookup Record Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Lookup record Action component. **Setup**: The Lookup Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the file that you want to be validated. In other words, select the file that will contain the data that you want to verify is valid.
Lookup Field	Select the field from this pull-down list that will provide validation for the value entered in the field selected from the Lookup Source pull-down list.

Note: UPG enables you to specify a hierarchical lookup, up to three levels deep. If a hierarchical lookup is defined, all values in the lookup targets must be validated against the respective lookup field for the value entered in the field selected from the Lookup Source pull-down list to be "valid."

Lookup Field 2

Select a field from this pull-down list that you want to perform the secondary lookup; this lookup validates the entry entered in the field specified in the Lookup Source 2 field.

Lookup Field 3

Select a field from this pull-down list that you want to perform the third level lookup; this lookup validates the entry entered in the field specified in the Lookup Source 3 field.

Lookup Source

The field selected from this pull-down list will be validated by the field selected from the Lookup Field pull-down list.

Lookup Source 2

If a Lookup Field 2 is defined, the file selected from this pull-down list will be validated by the field selected in the Lookup Field 2 pull-down list.

Lookup Source 3

If a Lookup Field 3 is defined, the file selected from this pull-down list will be validated by the field selected in the Lookup Field 3 pull-down list.

Action: Goto First Record (File)

Use the Goto First Record Action component to place the first record of the target file in the file buffer.

Field/Control	Description
Name	Enter the name of your Goto First Record Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Goto First Record Action component. **Setup**: The Goto First Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.
Key to Process	Select the key that you want to sort on. The record that is first according this sort order will be placed in your file buffer. If no Key to Process is selected, the first record in the file will be placed in the file buffer.

Action: Goto Last Record (File)

Use the Goto Last Record Action component to place the last record of the target file in the file buffer.

Field/Control	Description
Name	Enter the name of your Goto Last Record Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Goto Last Record Action component. **Setup**: The Goto Last Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.
Key to Process	Select the key that you want to sort on. The record that is first according this sort order will be placed in your file buffer. If no Key to Process is selected, the first record in the file will be placed in the file buffer.

Action: Goto Next Record (File)

Use the Goto Next Record Action component to place the next record of the target file in the file buffer.

Field/Control	Description
Name	Enter the name of your Goto Next Record Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Goto Next Record Action component. **Setup**: The Goto Next Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.

Action: Goto Previous Record (File)

Use the Goto Previous Record Action component to place the previous record of the target file in the file buffer.

Field/Control	Description
Name	Enter the name of your Goto Previous Record Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Goto Previous Record Action component. **Setup**: The Goto Previous Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.

Action: Does File Exist? (File)

The Does File Exist? Action component checks to see if the target file exists. If the target file does exist, a value of TRUE (non-zero) will be returned and if the target file does not exist, a value of FALSE (zero) will be returned. This Action component will be useful when combined with an IF or Loop action component and the Last Calculation (Numeric) expression.

Field/Control	Description
Name	Enter the name of your Does File Exist? Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Does File Exist Action component. **Setup**: The Does File Exist Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.

Action: Delete Record (File)

The Delete Record Action component deletes the record currently stored in the file buffer.

Field/Control	Description
Name	Enter the name of your Delete Record Value action Component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Delete Record Action component. **Setup**: The Delete Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.

Action: Change Record (File)

Saves any changes that have been made to the record in the file buffer. If you perform a lookup or use one of the record navigation Actions (i.e. Goto First Record) and you make a change to the data, use the Change Record Action component to save the change.

Field/Control	Description
Name	Enter the name of your Change Record Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Change Record Action Component. **Setup**: The Change Record Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.

Action: Record Count (File)

The Record Count Action component, as the name suggests, counts the records stored in the target file. This Action component will be useful when combined with an IF or Loop action component and the Last Calculation (Numeric) expression.

Field/Control	Description
Name	Enter the name of your Record Count Action component in the **Name** field. Remember, this control name must comply with C naming conventions.
When to Execute	Select the event that will trigger your Calculate Text Value Action Component. **Setup**: The Record Count Action component will be processed before your form is written to your screen. **Process**: Occurs at the end of a data collection sequence, before you return to the loop field. **Exit**: Occurs when you press the exit key to leave the current screen. **After Input Field**: Executes when the user presses <ENTER> or the input device sends the return character while in the specified field (see Input Field to Execute After below). **Called by Other Control**: Is run when another control calls the File Action Component.
File to Process	Select the target file for this action component.

C

APPENDIX - EXPRESSION BUILDER

Expressions

Expression	Description
Current Date	Returns the current date.
Current Time	Returns the current time.
Disk Space Free (Bytes)	Returns the available disk space on the current computer.
Enter Code Directly	Allows the user to enter a C condition statement; the value entered in here would be as directly entering the condition within C source code.
External Variable	Allows you to retrieve the value of an external variable; you will need to enter the appropriate C code to retrieve the value of the external variable.
Field/Variable	Returns the value contained in the specified field; the field will be selected based on the file.field combination.
Fixed Number (Float)	Allows you to specify a floating point fixed number in your expression. Floating point values may contain decimals and are also referred to as *real* numbers. Floating point values are declared as a double precision value in the generated source code.
Fixed Number (Integer)	Allows you to specify a fixed integer (whole number). An integer is stored as a long value in the generated source code. When possible, use an integer instead of a floating point number. This will allow your CPU to operate more efficiently (floating point values are more complex than integers).
Fixed Text	Allows you to enter a fixed text value into your condition statement or calculation. This particular expression will often be used as the second parameter in many of the different Text expressions.
Last Calculation (Text)	Returns the result of the last Calculate Text Value that was executed. This value can be returned even if a Calculate Numeric Value has occurred after the Calculate Text Value.
Last Calculation (Numerical)	Returns the result of the last Calculate Numeric Value that was executed. This value can be returned even if a Calculate Text Value has occurred after the Calculate Numeric Value.
Last Entry	Returns the value that was last entered into your portable data collection application.
Last Entry Control Name	Similar to the Last Entry expression, the Last Entry Control Name expression will identify, by name, which input field received the last data entry.
Last Data Entry Identifier	Returns the data identifier that was last encountered in your portable data collection application.
Last Entry from Keyboard	Returns the value that was last entered into your portable application via your portable's keyboard.

Table C-1

Expression	Description
Last Entry from Scanner	Returns the value that was last entered into your portable application via your portable's scanner (laser, wand, CCD, etc.).
Last Entry Length	Returns the length (in characters) of the value last entered into your portable application.
Last Entry Match Group	If you have multiple entry patterns defined for an input field, you can use the Last Entry Match Group expression to detect which entry pattern was used. For example, if you had defined ####|@@@@|##@@ in the Entry Pattern field in the Verify tab of an input field, and the user entered four (4) numbers, the entry pattern would return a value of 1 (since #### is the first entry pattern). If the user entered two (2) numbers followed by two (2) letters, the expression would return a value of 3 (since ##@@ is the third entry pattern).
Logical False	Sets the value to FALSE (this is useful for testing and debugging).
Logical True	Sets the value to TRUE (this is useful for testing and debugging).
Lookup Failed	Returns a value of TRUE (non-zero) if the last lookup failed.
Lookup Failed and User Override	Returns a value of TRUE (non-zero) if the last lookup failed, but the user accepted (overrided the lookup/verify) the value anyways.
File Missing	Returns a value of TRUE (non-zero) if the validation file was missing.
Lookup Succeeded	Returns a value of TRUE (non-zero) if the last lookup succeeded (the entered value was contained in the validation file).
Lookup Succeeded or File Missing	Returns a value of TRUE (non-zero) if the last lookup succeeded (the entered value was contained in the validation file) or the validation file was missing. This expression will only work properly if the Ignore if Missing checkbox is enabled in the Verify tab of the input field when you defined the lookup.
Lookup Succeeded or User Override	Returns a value of TRUE (non-zero) if the last lookup succeeded (the entered value was contained in the validation file) or the user accepted (overrided the lookup/verify) the value anyways.
Number Equal	Returns a value of TRUE if the two values defined in the First and Second Number pull-down lists are equal.
Number Greater	Returns a value of TRUE if the value defined in the First Number pull-down list is greater than the value selected in the Second Number pull-down lists.
Number Greater or Equal	Returns a value of TRUE if the value defined in the First Number pull-down list is greater than or equal to the value selected in the Second Number pull-down lists.
Number Less	Returns a value of TRUE if the value defined in the First Number pull-down list is less than the value selected in the Second Number pull-down lists.
Number Less or Equal	Returns a value of TRUE if the value defined in the First Number pull-down list is less than or equal to the value selected in the Second Number pull-down lists.

Table C-1

<table>
<tr><th>Expression</th><th>Description</th></tr>
<tr><td>Number Not Equal</td><td>Returns a value of TRUE if the two values defined in the First and Second Number pull-down lists are not equal.</td></tr>
<tr><td>Pattern Match</td><td>Use this expression to apply an entry pattern to any given field; specify the entry pattern in the same manner as described in Chapter 5, Data Verification. You might want to apply an entry pattern to values read from an INI file or from a validation file.</td></tr>
<tr><td colspan="2"><b>Note:</b> All Text expressions are based on the numeric value of the text. The numeric values are calculated on an alphabetic scale, where, for example, Z is greater than A. Another analogy might be to consider names entered in a phone book; Smith, Bob is less than Smith, John.</td></tr>
<tr><td>Text Compare</td><td>Compares two text values; this expression is case sensitive (a capital letter is greater than a lower case letter). This expression returns the following values:
• If value of the First Parameter is less than Second Parameter, a number less than zero (0) will be returned.
• If value of the First Parameter is equal to the Second Parameter, zero (0) will be returned.
• If value of the First Parameter is greater than the Second Parameter, a number greater than zero (0) will be returned.</td></tr>
<tr><td>Text Compare (No Case)</td><td>Compares two text values; this expression is not case sensitive (capital and lower case letters have the same value - a is equal to A). This expression returns the following values:
• If value of the First Parameter is less than Second Parameter, a number less than zero (0) will be returned.
• If value of the First Parameter is equal to the Second Parameter, zero (0) will be returned.
• If value of the First Parameter is greater than the Second Parameter, a number greater than zero (0) will be returned.</td></tr>
<tr><td>Text Equal</td><td>Returns a value of TRUE (non-zero) if the two values specified in the First and Second parameter pull-down lists are equal (this expression is case sensitive).</td></tr>
<tr><td>Text Equal (No Case)</td><td>Returns a value of TRUE (non-zero) if the two values specified in the First and Second parameter pull-down lists are equal (this expression is not case sensitive).</td></tr>
<tr><td>Text Greater</td><td>Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is greater than the value specified in the Second Parameter pull-down list (this expression is case sensitive).</td></tr>
<tr><td>Text Greater or Equal</td><td>Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is greater than or equal to the value specified in the Second Parameter pull-down list (this expression is case sensitive).</td></tr>
</table>

Table C-1

Expression	Description
Text Greater or Equal (No Case)	Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is greater than or equal to the value specified in the Second Parameter pull-down list (this expression is not case sensitive).
Text Greater (No Case)	Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is greater than the value specified in the Second Parameter pull-down list (this expression is not case sensitive).
Text Instring	Tells you if a substring is found in another string. Returns 0 if not and the position in the target string if so.
Text Instring (No Case)	Tells you if a substring is found in another string. Returns 0 if not and the position in the target string if so (ignores case).
Text Less	Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is less than the value specified in the Second Parameter pull-down list (this expression is case sensitive).
Text Less or Equal	Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is less than or equal to the value specified in the Second Parameter pull-down list (this expression is case sensitive).
Text Less or Equal (No Case)	Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is less than the value specified in the Second Parameter pull-down list (this expression is not case sensitive).
Text Less (No Case)	Returns a value of TRUE (non-zero) if the value specified in the First Parameter pull-down list is less than or equal to the value specified in the Second Parameter pull-down list (this expression is not case sensitive).
Text Lower Case	Converts value selected in the Text to Convert pull-down list to all lower-case letters.
Text Not Equal	Returns a value of TRUE (non-zero) if the two values specified in the First and Second parameter pull-down lists are not equal (this expression is case sensitive).
Text Not Equal (No Case)	Returns a value of TRUE (non-zero) if the two values specified in the First and Second parameter pull-down lists are not equal (this expression is not case sensitive).
Text Substring	Returns a portion of a text field, e.g. substring("Testing",1,2) returns "Te."
Text Upper Case	Converts value selected in the Text to Convert pull-down list to all upper-case letters.

Table C-1

Operators

Operator	Symbol	Description
Add	+	Addition
Subtract	-	Subtraction
Multiply	*	Multiplication
Divide	/	Division
Open Parenthesis	(	Must be used with a Close Parenthesis to establish order of operations.
Close Parenthesis	)	Must be used with an Open Parenthesis to establish order of operations.
Mod	%	Modulus (remainder operator)
Increment	++	Postfix Increment
Decrement	--	Postfix Decrement
Add To	+=	Addition and Assignment
Multiply By	*=	Multiplication and Assignment
Divide By	/=	Division and Assignment
Logical Less	<	Test for less than
Logical Less or Equal	<=	Test for less than or equal to
Logical Equal	==	Test for equality
Logical Not Equal	|=	Test for inequality
Logical Greater or Equal	>=	Test for greater than or equal to
Logical Greater	>	Test for greater than
Logical OR	||	Logical OR
Logical AND	&&	Logical AND
Logical NOT	!	Logical NOT
Bitwise OR	|	Bitwise OR
Bitwise AND	&	Bitwise AND

Operator	Symbol	Description
Bitwise NOT	~	Bitwise NOT

Table C-2

D

APPENDIX - UPG INI FILES

Overview of UPG INI Files

While you have the capability of creating your own INI files to be used in your portable applications, there are two automatically generated INI files that you need to be aware of. Knowing how these INI files work will greatly increase the power and flexibility of your portable data collection applications.

The UPG.INI file contains the settings for your portable's hardware specific requirements; this particular file also contains several settings that affect your portable application, such as the time and date formats and which communications utility to use (xfer.exe or upgxfer.exe).

The application.ini file is specific to your portable application. If your program executable is named INVEN.EXE, then your application.ini will be INVEN.INI. This INI file contains information about your individual data collection sequences and the parameters of the individual input fields. The developer can also create an APPLICATIONP.INI file for user definable prompts (prompt substitution).

UPG.INI File

The UPG.INI file contains the settings that control the way that your portable data collection unit interacts with your UPG generated application.

```
[General]
Screen Rows=16
Screen Cols=20
Upper Case Input=0
Upper Case Display=0
Quit On Ctrl-C=1
Number Keys on Menus=1
Hide Cursor=0
Symbol Mode=0
Escape Key Name=ESC
Date Format=%m/%d/%y
Time Format=%H:%M:%S
Yes Keys=+1Y
No Keys=-0N
Yes Text=Y
No Text=N
Language=1
```

```
XFer Program=UPGXFer.exe
Direct Port=0
Direct XModem=0
Direct Baud=38400
Modem Port=0
Modem XModem=0
Modem Baud=0
Modem Dial=
Modem Answer=
XFer Shell=0
Restart Timeout=0
True Text=True
False Text=False
Modem Phone=333-444-4432
```

Screen Rows

Specifies the height of your portable display in rows. When you select a target portable from the Portable menu, the proper display height is written to this entry.

Screen Cols

Specifies the width of your portable display in columns. When you select a target portable from the Portablemenu,theproperdisplaywidthiswrittentothisentry.

Upper Case Input

Set to 1 to force all upper case input. Set to 0 (default) to leave case as entered.

Quit On Ctrl-C

Set to 1 (default when running in a Windows DOS box) to allow user to press <CTRL><C> to exit to DOS. Set to 0 (default on portable) to disable.

Number Keys on Menus

Specifies whether or not the number keys on your portable can function as F keys when no data input is required. For example, if your main menu had an F8 hotkey, instead of pressing <F8> on your portable, you can simply press 8 on you number pad to execute the F8 feature.

Enter 1 in this entry (default) to enable your number pad to function as F keys. Enter 0 in this entry to disable this feature.

Symbol Mode

Set to 1 to turn on special library needed for Symbol 3xxx portables. Set to 0 (default) to disable special Symbol library. (This entry will default to 1, however, when a Symbol 3xxx is selected in the Portable menu.)

Escape Key Name

Name of the escape key used on specific portable.

Percon FalconESC
SymbolCLR
Intermec-JanusESC

Date Format

Sets the date format used within your portable application (affects both entered values and time stamping). UPG defaults to the Standard US date format (`%m/%d/%y`). If you want to use the European format, change this setting to `%d/%m/%y` (this format will handle a 2 and 4 digit year).

Time Format

Sets the time format used with your portable application (affects both entered values and time stamping). UPG defaults to `%H:%M:%S`. If you would like to exclude the seconds from the time format, simply edit this entry to read `%H:%M`.

Yes Keys

Specify which keys can be pressed to enter a Yes value at a Yes/No prompt. UPG defaults to `+1Y`, which means the user can either press the <+>, <1>, or <Y> keys to enter a Yes value. Edit the *Yes Keys* entry to modify which keys can be pressed for a Yes value.

No Keys

Specify which keys can be pressed to enter a No value at a Yes/No prompt. UPG defaults to `-0N`, which means the user can either press the <->, <0>, or <N> keys to enter a No value. Edit the *No Keys* entry to modify which keys can be pressed for a No value.

Language

Sets the current language for your portable application. UPG defaults to 1 (English). If you have built your portable application to support multiple languages, you will need to enter the proper language value for your portable application. Please enter a language value based on the following values:

English1
French2
German3
Spanish4

XFer Program

Specifies which communications utility will be used for transferring files between the portable and host PC. UPG defaults to UPGXfer. You may change the XFer Program entry according to the following values:

UPGXfer Supports ZModem.

XFerSupports XModem and ACK/NAK (Percon proprietary protocol to be used with existing Percon file transfer utilities that support ACK/NAK).

Direct Port

Sets the COM port to be used for direct connect file transfer sessions between the portable and host. Specify the COM port with the following syntax:

-p[*COM port*]

For example, if you wanted to set the COM port to COM port 2, you would edit the Direct Port entry to read:

```
-p2
```

Direct XModem

Set to 1 to use XModem as the direct connection transfer protocol. Set to 0 (default) to leave protocol set to ZModem.

Direct Baud

Sets the baud rate for file direct connect transfer sessions between the portable and host PC. UPG defaults to the most reliable transfer rate based on the target portable selected in the Portable menu. If you modify this entry, check your portable hardware documentation for supported baud rates. Specify the baud rate with the following syntax:

-b[*baud rate*]

For example, if you wanted to set the baud rate to 38,400, you would edit the Direct Baud entry to read:

`-b38400`.

Modem Port

Sets the COM port to be used for modem file transfer sessions between the portable and host. Specify the COM port with the following syntax:

-p[*COM port*]

For example, if you wanted set the COM port to COM port 2, you would edit the Modem Port entry to read:

`-p2`

Modem XModem

Set to 1 to use XModem as the modem connection transfer protocol. Set to 0 (default) to leave protocol set to ZModem.

Modem Baud

Sets the baud rate for file modem transfer sessions between the portable and host PC. UPG defaults to the most reliable transfer rate based on the target portable selected in the Portable menu. If you modify this entry, check your portable hardware documentation for supported baud rates. Specify the baud rate with the following syntax:

-b[*baud rate*]

For example, if you wanted to set the baud rate to 9600, you would edit the Modem Baud entry to read:

`-b9600`.

Modem Dial
Enter the dialing string (using standard AT commands) that your portable application should use to initiate a modem connection with a host PC. Defaults to ATDT.

Modem Answer
Enter the answer string (using standard AT commands) that your portable application to receive a modem connection with a host PC. Defaults to ATS0=2.

XFer Shell
Set to 1 to run your transfer session in a shell (your portable application will be closed, your transfer session executed, and then the portable application will be reopened - this will free up resources when working with large applications and/or files). Set to 0 (default) to run transfer sessions from within your portable application.

Restart Timeout
UPG gives the you the ability to automatically restart your application after a specified amount of portable inactivity. Use the Restart Timeout entry to specify this amount of time (in seconds). For example, if you wanted your portable application to restart after 5 minutes of portable inactivity, you would enter `300`.

True Text
Enter the text that you want shown for a true value in a Yes/No field. Defaults to “YES.”

False Text
Enter the text that you want shown for a false value in a Yes/No field. Defaults to “NO.”

Modem Phone
Enter the phone number that will be dialed when using a modem to initiate a communications session with a host PC.

APPLICATION.INI File

The application.ini file gives you the ability to modify the way your portable application operates. Specifically, you can change the following:

- ❑ Name and path of data files
- ❑ Order and behavior of input fields
- ❑ Modify the state of available persistent values

```
[FileNames]                                      Data File
collect=collect.txt

[Inven_Collect_Data]                             Multipage Sequence
S=1,2,3
1=4,25,Warehouse A
2=4,25,,,####@@|@@@######
3=5,25,,P,,abcdefghijklmnopqrstuvwxyz

[Persistent]                                     Persistent Settings
Inven_Collect_Data.Entry03.Prop_1=1
```

You can modify these settings with a simple text editor, or you can build an INI file structure within UPG to simulate the application.ini file. If you simulate the file structure, you will be able to modify the INI settings directly from within your UPG generated application (please see Chapter 6, Beyond the Basics for information on building an INI file structure to simulate the application.ini file).

Data File

You may change the name and the location of the data files for your portable application.

[tool name] = [path and file]

Tool Name — This is the name of the tool that defines the data file that you want to modify. Be certain not to modify this value, as any change may cause errors when trying to rename or relocate the data file.

Path and File — Specifies the DOS file name that collected data is written to. If necessary, you may specify a new name for this data file; any collected data will be written to the file specified in the INI file. Also, you may include a path to save the data to a file located in a directory different than the where your portable application resides.

Multipage Sequence

There are two pieces to the multipage sequence entries in the application.ini file. The first piece (i.e. S=1,2,3) denotes the sequence of data entry. The second piece modifies the behavior of the individual input fields.

Data Entry Order

```
S = [sequence order]
```

S

Fixed parameter that denotes input field sequence order; this listing will always appear directly below the multipage sequence tool name (i.e. [Inven_Collect_Data]).

Sequence Order

Lists the order that the input fields accept data entry. The default sequence is the order that input fields appear in the multipage sequence (open the multipage sequence in the UPG work environment to view the original input field order). You can rearrange the sequence to customize that input field sequence.

For example, if Entry01, Entry02, and Entry03 are labeled as 1, 2, and 3 respectively, you can do the following: S=2,3,1. Your new data entry sequence will be Entry02, Entry03, followed by Entry01.

Input Field Behavior

```
[Input Field #] = [Configuration Parameters],[max length],[default],
                [data identifer],[entry pattern],[valid characters]
```

Input Field #

The series number of the input field; these numbers are listed in the order that the input fields are listed in. For example, if you had Entry01 followed by Entry02 listed on the first page of your multipage sequence, they would be numbered 1 and 2 respectively.

Configuration Parameter

Modifies the behavior of the input field. To specify compound configuration parameters, add the individual parameter numbers together.

Input Disabled	1
Input Required (may reuse existing value)	2
Input Cannot be Blank	4
Input Must be in File if Non-blank	8
Input is Display Only	16
Force Upper Case	32
Force Lower Case	64
Special Case Used in `fieldpropinit`	128
Input Must be From Scanner	256
Input Must be From Keyboard	512
Data Identifier is Required	1024
Re-use Default Value Each Time	2048
Don’t Show User Entry on Screen (echo *)	4096

For example, if you wanted to specify that data entry must be from the scanner, a data identifier is required, and that an input is required, the configuration parameter would be set to `1282`.

Max Length

Shows the max length that the input field will accept as data entry. You may modify the max length up to twice the original setting.

Default

Specifies a default value for this input field. Any entry in this parameter will be visible when the user comes to this input field. ENTER can be pressed to accept the default or the user can modify the value as necessary.

Data Identifier

Specifies any data identifier for this input field. If you want to force the data identifier to be included in the data entry, either enable the Require Data Identifier checkbox in the Options tab of an Input Field definition screen or be sure to include the Data Identifier is

Required configuration parameter (400). See Configuration Parameters above for more information.

Entry Pattern Specifies any entry pattern for this input field. Modifications can be made to the INI file in the same manner as entry patterns are defined in the Entry Pattern field in the Verify tab of an Input Field definition screen (see Chapter 5, Data Verification for more information).

Valid Characters Lists the set of valid characters for this input field. Any character entered that is not in the valid character set will immediately be rejected (see Chapter 5, Data Verification for more information).

APPLICATIONP.INI File

The last INI file that needs to be discussed here is the applicationP.INI file (if your application name is INVEN.EXE, your applicationP.INI file will be INVENP.INI). This INI file is not generated by UPG, but rather needs to be assembled by you. In short, the applicationP.INI file allows you to substitute the display text of each of your user interface controls (i.e. input fields, hotkeys, etc.). This is a powerful feature that will let your users customize the portable data collection application to meet their individual needs.

Simulating the applicationP.INI file within your UPG portable application will let your users change the user interface display text from within the UPG generated application, without ever having to edit an INI file.

The structure of the applicationP.INI file is quite simple.

[control name]=[substitute display text]

```
CollectItemNo=Item No:
CollectSite=Site:
CollectLoc=Location:
CollectQty=Quantity:
```

All controls that share the control name, will have the same substitution occur. This sharing of substitute display text makes the changes in your application global. For example, you might have several data collection sequences that have an input field for Site and the user may want all instances of Site to be Building. Defining the same control name between different forms will make this global assignment much easier, in terms of development.

With this flexibility, however, comes a certain responsibility. You need to make sure that a global replacement is acceptable. If you need to ensure that each display text is substituted on an individual basis, you will need to specify unique control names between all of the user interface controls.

E

Appendix - Portable Information

Percon Falcon

	Manufacturer Information	UPG Notes
Drive A:	Contains COMMAND.COM; used to boot portable unit. Not user accessible.	Not modified by UPG.
Drive B:	Contains system utilities and device drivers. Not user accessible.	Not modified by UPG.
Drive C:	Flash ROM; typically used for applications and data utilities. User accessible. Size: 720K Available: 720K	Your UPG application will be installed onto the C drive when you program your Falcon unit. Your collected data will be saved to the C drive by default.
Drive D:	Data Drive (RAM disk). This is where your portable application will save the collected data. Size: 1MB Available: 1MB	If specified in your UPG application, your UPG application can save the collected data to the D drive.
Programming Mode Key Sequence	Type `LD` at any DOS prompt.	You will need to use the Programming Mode Key Sequence to ready your Percon Falcon unit to receive your UPG generated application.
Reboot Key	<FN 2><ALT><POWER>	
Programming Utility	XFER.EXE	UPG uses XFER.EXE to program your Percon Falcon with the UPG generated application.
Data Transfer Utility		UPGXFER.EXE is used to transfer data between Host PC and portable using ZModem (default). Use XFER.EXE to support XModem.
Percon Tech Support	1-800-929-7899	
Percon WWW Site	http://www.percon.com	

Symbol 3100 and 3805

	Manufacturer Information	UPG Notes
Drive A:	Contains the default operating system (emulated by EEPROM). (BIOS)	Not modified by UPG.
Drive B:	User area of NVM (Non-Volatile Memory). Where you “burn in” or place your HEX image. Size: 256K	Your UPG application will be installed onto the B drive when you program your Symbol unit.
Drive C:	Not Available	
Drive D:	Data Drive (RAM disk). This is where your portable application will save the collected data. (Available disk space varies with model.)	This is where your UPG generated application will save the collected data to, as well as store your application’s configuration (*.INI) files.
Programming Mode Key Sequence		You will need to use the Programming Mode Key Sequence to ready your Symbol unit to receive your UPG generated application.
Symbol Model	**Key Sequence**	
3100 (35 key)	Loader Mode: <BKSP><Shift><Power> Cold Boot: <Space><FUNC><Up Arrow><Power>	
3100 (47 key)	Loader Mode: <F><I><Power> Cold Boot: <A><B><D><Power>	
3805 (35 key)	Loader Mode: <BKSP><Shift><Power> Cold Boot: <Space><FUNC><Up Arrow><Power>	
3805 (46 key)	Loader Mode: <F><I><Power> Cold Boot: <A><B><D><Power>	
Programming Utility	SENDHEX.EXE	UPG uses the Symbol SENDHEX.EXE to program your portable with the UPG generated application.
Data Transfer Utility		UPGXFER.EXE will be used on the Host PC and XFERSMBL.EXE will be used on the portable to transfer data between Host PC and portable.
Symbol Tech Support	1-800-653-5350	
Symbol WWW Site	http://www.symbol.com	

Intermec-Janus 2010/2020

	Manufacturer Information	UPG Notes
Drive C:	User Files. Contains AUTOEXEC.BAT, PC Card Drivers, IRL Files.	Your UPG application will be installed onto the C drive when you program your Intermec-Janus unit.
Drive D:	Operating system; DOS files (it is recommended that you do not modify the contents of this drive). Size: 512K Available: 256K	Not modified by UPG.
Drive E:	Data drive (typically where you would write your collected data to). Size: 256K Available: 256K	This is where your UPG generated application will save the collected data to, as well as store your application's configuration (*.INI) files.
Programming Mode Key Sequence	1 Turn unit off. 2 Press <F2>, <LEFT ARROW>, and <2>, then release. 3 Press <2> then release. 4 Press the Power button.	You will need to use the Programming Mode Key Sequence to ready your Intermec-Janus unit to receive your UPG generated application.
Programming Utility	LOADER.EXE	UPG uses the Intermec-Janus LOADER.EXE to program your portable with the UPG generated application.
Data Transfer Utility		UPGXFER.EXE is used to transfer data between Host PC and portable.
Intermec-Janus Tech Support	1-800-755-5505	
Intermec WWW Site	http://www.intermec.com	

F

APPENDIX - XMODEM VS. ZMODEM

UPG Communications: A Developer’s Discussion

Percon’s Universal Program Generator gives you the ability to build powerful portable data collection applications quickly and easily, including integrated communications capabilities. UPG development allows you, as a developer, to use both the XModem and ZModem protocols for your communications needs.

Specifically, UPG includes the following three transfer utilities:

❑ XFER.EXE
❑ UPGXFER.EXE
❑ XFERSMBL.EXE

XFER.EXE allows you to use XModem to communicate between your Host PC and your Percon Falcon. By default, XFER.EXE is used to program the Percon Falcon. XFER.EXE, however, can also be used as the communications utility to transfer data files between the Host PC and Falcon. If XFER.EXE is used as the communications utility, your portable users will be able to communicate with existing Percon file transfer utilities, including the Falcon Configuration Utility, without the need to purchase UPG Runtime.

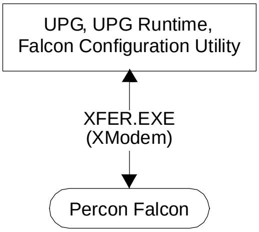


Figure F-1: XFER.EXE supports an "open" communications arrangement.

UPGXFER.EXE utilizes the ZModem protocol to initiate communications between a portable (works with all supported portables) and the Host PC. In addition to simple direct connect data transfer sessions, you may also communicate between your Host PC and portable via modem (using the standard AT command set). UPGXFER.EXE, unlike XFER.EXE, also supports "Host" mode file transfer sessions (a PC or portable in "Host" mode will continue to receive files until the transfer session is terminated). It is required that the Host PC use a licensed copy of UPG Runtime to communicate with portables using UPGXFER.EXE (ZModem).

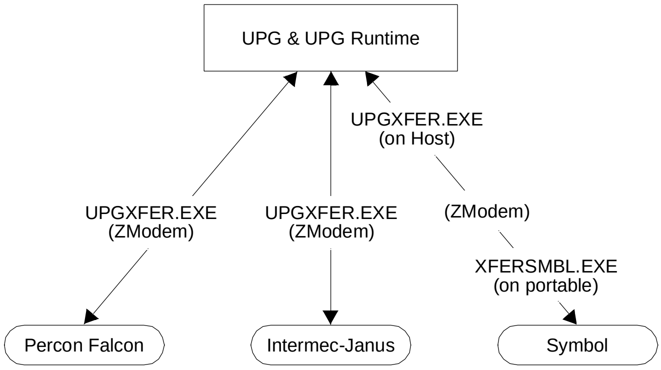


Figure F-2: UPGXFER.EXE will enforce a UPG Runtime communications requirement on the Host PC for file transfer communications.

XFERSMBL.EXE is a customized version of UPGXFER.EXE that has been designed to meet the special requirements of a Symbol portable data collection unit. This communication utility contains the same feature set and operates the same as UPGXFER.EXE. XFERSMBL.EXE is loaded to the portable unit when you program a Symbol unit. The Host PC utilizes UPGXFER.EXE during communications sessions with a Symbol unit, while the Symbol unit uses XFERSMBL.EXE.

A common concern among portable data collection application developers is how to enforce a runtime communications utility requirement on the Host PC. This requirement will of course help resellers maintain a level of recurring revenue, by selling the required runtime communications utility. There are, however, some opponents to this position who would rather have a royalty free portable application.

UPG addresses both sides of this issue. UPG developers who want to build a royalty free runtime (an application that does not require UPG Runtime), can elect to use XFER.EXE to perform communications functions with the Percon Falcon. The use of XFER.EXE allows users of the portable application to communicate with their host using the Falcon Configuration Utility (ships with every purchase of the Percon Falcon). To specify XFER.EXE as the communications utility, select "XModem" from the **Communications Setup** form in UPG.

On the flip side, developers who want to require a runtime application, should use UPGXFER.EXE as the communications utility. This will require each user of your portable application to use UPG Runtime on their Host PC to communicate with their portable application. To specify UPGXFER.EXE as the communications utility, select "ZModem" from the **Communications Setup** form in UPG.